WebSphere® IBM WebSphere Process Server for Multiplatforms
Version 7.0.0

Développement et déploiement de modules

IBM®

WebSphere® IBM WebSphere Process Server for Multiplatforms
Version 7.0.0

Développement et déploiement de modules

IBM®

Remarque
Certaines illustrations de ce manuel ne sont pas disponibles en français à la date d'édition.

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE.

Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. Les informations qui y sont fournies sont susceptibles d'être modifiées avant que les produits décrits ne deviennent eux-mêmes disponibles. En outre, il peut contenir des informations ou des références concernant certains produits, logiciels ou services non annoncés dans ce pays. Cela ne signifie cependant pas qu'ils y seront annoncés.

Pour plus de détails, pour toute demande d'ordre technique, ou pour obtenir des exemplaires de documents IBM, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial.

Vous pouvez également consulter les serveurs Internet suivants :

- http://www.fr.ibm.com (serveur IBM en France)
- http://www.can.ibm.com (serveur IBM au Canada)
- http://www.ibm.com (serveur IBM aux Etats-Unis)

Compagnie IBM France
17 avenue de l'Europe
92275 Bois-Colombes Cedex

Table des matières

Tableaux

Partie 1. Développement d'applications

Chapitre 1. Développement de solutions de gestion des processus métier (BPM)

Cette section évoque les principes du modèle de programmation de la gestion des processus métier (BPM). Elle présente l'architecture SCA (Service Component Architecture) et les modèles associés à l'intégration métier.

BPM est la discipline permettant aux entreprise d'identifier, consolider et optimiser des processus métier. Son objectif est d'améliorer la productivité et maximiser l'efficacité de l'entreprise. L'intérêt grandissant que suscite BPM est dû aux fusions et aux consolidations d'entreprises et au fait qu'elles développent des bibliothèques d'éléments d'actif informationnel divers. Ces éléments d'actif manquent souvent de cohérence et de coordination, ce qui crée des "îlots d'informations".

BPM estvétroitement lié à l'architecture orientée services (SOA). En fonction du type d'entreprise et de l'étendue des besoins d'intégration, BMP impose diverses exigences aux services informatiques. Certains projets sont seulement confrontés à quelques aspects de ces exigences, alors que des projets de plus grande envergure regroupent un grand nombre d'entre elles. Vous trouverez ci-dessous quelques aspects, parmi les plus courants, composant des projets BPM :

- **Intégration d'applications** est une exigence courante. La complexité des projets d'intégration d'applications varie selon qu'il s'agit de situations simples, dans lesquelles vous devez garantir qu'un nombre réduit d'applications peut partager des informations, ou de situations plus complexes, dans lesquelles des transactions et des échanges de données doivent apparaître simultanément dans plusieurs applications dorsales. L'intégration d'applications complexe exige souvent une gestion de l'unité de travail compliquée, mais aussi de la transformation et du mappage.
- **Automatisation des processus** est un autre aspect clé qui assure que les activités exercées par une personne ou une entreprise déclenchent automatiquement des conséquences ailleurs. Ceci garantit l'accomplissement du processus métier global. Par exemple, lorsqu'une entreprise embauche un employé, les informations de la feuille de paie doivent être mises à jour, le service de sécurité doit appliquer des actions adéquates, les outils requis doivent être mis à la disposition de l'employé, etc. Certaines activités composant un processus peuvent capturer les entrées des utilisateurs et leur interaction, alors que d'autres peuvent appeler des scripts sur des systèmes dorsaux et d'autres services présents dans l'environnement.
- **Connectivité** est un aspect abstrait, et pourtant essentiel, pour une entreprise et les partenaires commerciaux. La connectivité fait référence au flux d'informations échangées entre les entreprises ou les sociétés et la capacité à accéder à des services informatiques distribués.

Certains défis techniques liés aux implémentations d'intégrations métier peuvent se résumer de la façon suivante :

- Traiter différents formats de données et ne pas être en mesure d'effectuer une transformation efficace des données
- Traiter différents protocoles et mécanismes pour accéder à des services informatiques qui ont été développés via des technologies différentes
- Organiser différents services informatiques qui peuvent être distribués géographiquement ou offerts par différentes entreprises

- Fournir des règles et des mécanismes pour classer et gérer les services qui sont disponibles (gouvernance)

En tant que tel, BMP regroupe de nombreux thèmes et éléments qui sont également communs à l'architecture SOA. La vision d'IBM concernant BPM se fonde sur de nombreux concepts de base identiques figurant dans l'architecture SOA. L'une des conséquences directes de cette vision est que les solutions BPM exigent le recours à divers produits pour leur élaboration. IBM® fournit toute une gamme d'outils et de plateformes d'exécution afin de prendre en charge les différentes étapes et aspects opérationnels.

Pour paraphraser la vision d'IBM concernant BPM, il permet aux entreprises de définir, créer, fusionner, consolider et simplifier les processus métier à l'aide d'applications exécutées sous une infrastructure informatique SOA. Le travail de BPM se base véritablement sur des rôles. Au niveau macro, ceci implique la conception, le développement, la gouvernance, la gestion et la surveillance des applications de processus métier. Grâce à l'utilisation d'outils et de procédures adéquats, vous pouvez automatiser les processus métier impliquant des personnes et des systèmes hétérogènes à l'intérieur, mais aussi à l'extérieur, de l'entreprise. L'un des points clés de BPM est la possibilité d'optimiser vos activités commerciales afin qu'elles soient suffisamment efficaces, évolutives, fiables et flexibles pour gérer des modifications.

BPM exige des outils de développement, des serveurs d'exécution, des outils de surveillance, un référentiel de services, des boîtes à outils et des modèles de processus. Etant donné que BPM se compose de nombreux aspects différents, vous allez découvrir que plusieurs outils de développement doivent être utilisés pour développer une solution. Ces outils permettent aux développeurs d'intégration d'assembler des solutions métier complexes. Un serveur est un moteur d'activités à hautes performances ou un conteneur de services qui exécute des applications complexes. La direction veut toujours être informée de l'attribution des tâches au sein de l'entreprise et c'est à ce moment-là que les outils de surveillance interviennent. A mesure que les entreprises créent des processus ou services métier, la gouvernance, la classification et le stockage de ces services deviennent essentiels. Cette fonction est mise à disposition par un référentiel de services. Des boîtes à outils spéciales permettant de créer des éléments spécifiques à la solution, comme les connecteurs ou les adaptateurs de systèmes existants, sont souvent requises.

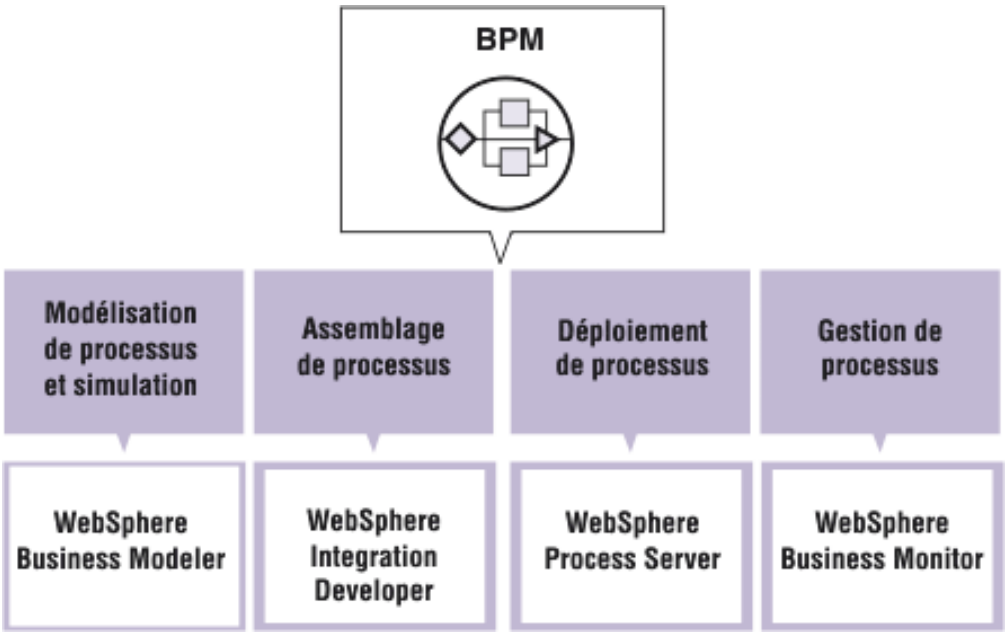


Figure 1. Les outils IBM prolongent le cycle de vie complet de la gestion des processus métier, ce qui permet de concevoir, assembler, déployer et gérer vos processus.

BPM ne se base pas sur un produit unique. Il mobilise la quasi-totalité du personnel et tous les aspects commerciaux d'une et de plusieurs entreprises. BPM regroupe de nombreux services et éléments figurant dans l'architecture de référence SOA.

Pour plus de détails sur ces concepts et pour consulter des exemples de programmation, reportez-vous à :

- *WebSphere Business Integration Primer : Process Server, BPEL, SCA, and SOA*, IBM Press, 2008.
- *Getting Started with IBM WebSphere Process Server and IBM WebSphere Enterprise Service Bus Part 1 : Development*, IBM Redbooks, SG24-7608-00, June 2008.
- *IBM Business Process Management Reviewer's Guide*, IBM Redpapers, REDP-4433-01, April 2009.

Concepts associés

«Architecture et modèles d'intégration métier»
Un projet de gestion métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Architecture et modèles d'intégration métier

Un projet de gestion métier classique implique la coordination de plusieurs actifs informatiques différents qui peuvent être en cours d'exécution sur différentes plateformes et qui ont été développés à différentes périodes via diverses technologies. La capacité à manipuler et échanger facilement des informations à

l'aide d'un ensemble de composants divers représente un défi technique énorme. Le modèle de programmation utilisé pour développer des solutions d'intégration métier permet de relever ce défi.

Cette section présente l'architecture SCA (Service Component Architecture) et les modèles associés à l'intégration métier. L'utilisation de modèles semblent se généraliser dans notre vie quotidienne. Les modèles de patrons, les modèles d'apprentissage personnalisé ("Think-and-learn") destinés aux enfants, les modèles pour la construction de maisons individuelles, les modèles de sculpture sur bois, les modèles de vols, les modèles de configuration des vents, les modèles de pratique en médecine, les modèles d'achat des clients, les modèles de flux de travaux, les modèles de conception en informatique, et beaucoup d'autres encore.

Les modèles se révèlent être utiles pour les concepteurs et développeurs de solutions. Il n'est donc pas surprenant de voir des modèles d'intégration métier et d'intégration d'entreprise. Il existe toute une gamme de modèles pouvant être appliqués à l'intégration métier, y compris des modèles de demande et de réponse pour le routage, des modèles de canaux (publication/abonnement) et beaucoup d'autres encore. Les modèles abstraits fournissent un modèle de résolution appliqués à une certaine catégorie de problèmes, alors que les modèles concrets fournissent des indications plus précises concernant la méthode d'implémentation d'une solution spécifique. Cette section traite des modèles devant traiter l'appel de données et de services qui constituent la structure du modèle de programmation de la stratégie des logiciels IBM pour l'intégration métier de WebSphere.

Concepts associés

«Scénarios d'intégration métier»
Les entreprises disposent de nombreux systèmes logiciels différents qui sont utilisés dans le cadre de leur activité. De plus, l'intégration de ces composants métier est propre à chaque entreprise.

«Rôles, produits et défis techniques», à la page 7
La réussite de projets d'intégration métier dépend de l'association de rôles de développement spécialisés, de techniques de programmation et de suites d'outils.

Scénarios d'intégration métier

Les entreprises disposent de nombreux systèmes logiciels différents qui sont utilisés dans le cadre de leur activité. De plus, l'intégration de ces composants métier est propre à chaque entreprise.

Les deux scénarios d'intégration métier les plus répandus sont les suivants :

- **Courtier d'intégration** : Dans ce scénario, la solution d'intégration métier agit comme intermédiaire entre diverses applications dorsales. Par exemple, vous devez vous assurer que lorsqu'un client passe une commande à l'aide de l'application de gestion de commandes en ligne, la transaction met à jour les informations correspondantes dans votre application dorsale CRM (Customer Relationship Management). Dans ce scénario, la solution d'intégration doit pouvoir capturer et éventuellement transformer les informations requises de l'application de gestion de commandes et appeler les services correspondants dans l'application CRM.
- **Automatisation des processus** : Dans ce scénario, la solution d'intégration sert de lien entre les différents services informatiques qui, dans le cas contraire, n'auraient aucun point commun. Par exemple, lorsqu'une entreprise embauche un employé, la série d'actions suivante doit se dérouler :
  - Les informations relatives à l'employé sont ajoutées au système de feuille de paie.

– L'employé doit pouvoir accéder physiquement aux infrastructures et un badge doit lui être fourni.
– L'entreprise peut fournir un ensemble de ressources matérielles à l'employé (espace bureau, ordinateur, etc.).
– Le service informatique doit créer un profil utilisateur pour l'employé et autoriser l'accès à toute une série d'applications.

  L'automatisation de ce processus est courante dans un scénario d'intégration métier. Dans ce cas, la solution implémente un flux automatisé qui est déclenché du fait de l'ajout de l'employé au système de feuille de paie. Ensuite, le flux déclenche les étapes suivantes en créant des éléments de travail pour les preneurs de décisions ou en appelant les services correspondants.

Dans ces deux scénarios, la solution d'intégration doit :

1. Utiliser des sources d'informations diverses et des formats de données différents, mais aussi pouvoir convertir des informations entre différents formats.
2. Pouvoir appeler divers services, en utilisant éventuellement différents mécanismes et protocoles d'appel.

Rôles, produits et défis techniques

La réussite de projets d'intégration métier dépend de l'association de rôles de développement spécialisés, de techniques de programmation et de suites d'outils.

Les projets d'intégration métier exigent quelques éléments de base :

- Une séparation claire des rôles au sein de l'entreprise en charge du développement, afin de favoriser la spécialisation, ce qui améliore généralement la qualité des composants individuels qui sont développés.
- Un modèle d'objet métier (BO - Business Object) commun qui permet aux informations métier d'être représentées dans un modèle logique commun.
- Un modèle de programmation qui sépare clairement les interfaces des implémentations, qui prend en charge un mécanisme d'appel de service générique totalement indépendant de l'implémentation et qui concerne uniquement les interfaces.
- Un ensemble d'outils et de produits intégrés qui prend en charge les rôles de développement et empêche leur séparation.

Les sections suivantes détaillent chacun de ces éléments.

Séparation claire des rôles

Un projet d'intégration métier a besoin de personnel pour quatre rôles de collaboration clairement distincts :

- **Analyste métier** : Il s'agit d'experts de domaine en charge de capturer les aspects métier d'un processus et de créer un modèle de processus qui représente correctement le processus même. Leur mission est d'optimiser les performances financières d'un processus. Ils ne s'intéressent pas aux aspects techniques de l'implémentation de processus.
- **Développeur de composants** : Ils sont en charge de l'implémentation de services et de composants individuels. Leur mission consiste à utiliser une technologie spécifique pour l'implémentation. Ce rôle exige une formation solide en programmation.

- **Spécialiste en intégration** : Ce rôle relativement nouveau consiste à assembler un ensemble de composants existants dans une solution d'intégration métier plus grande. Les développeurs d'intégration n'ont pas besoin de connaître les détails techniques de chaque composant et service qu'ils réutilisent et connectent entre eux. Théoriquement, ils doivent uniquement s'intéresser à comprendre les interfaces des services qu'ils assemblent. Ils doivent utiliser les outils d'intégration pour le processus d'assemblage.
- **Déployeur de solutions** : Les déployeurs et les administrateurs de solutions se chargent de rendre les solutions d'intégration métier disponibles aux utilisateurs finaux. En théorie, un déployeur de solutions se charge principalement de lier une solution aux ressources physiques prêtes à la faire fonctionner (bases de données, gestionnaires de files d'attente, etc.) et non pas de comprendre le fonctionnement interne d'une solution. Sa mission première est la qualité de service (QoS - Quality of Service).

Modèle d'objet métier commun

Comme nous l'avons mentionné précédemment, les aspects clés d'un projet d'intégration métier incluent la capacité à coordonner l'appel de plusieurs composants et à gérer l'échange de données entre eux. Plus particulièrement, différents composants peuvent utiliser différentes techniques pour représenter des éléments métier, comme les données d'une commande, les informations d'un client, etc. Par exemple, il se peut que vous ayez à intégrer une application Java™ qui utilise des EJB (Enterprise Java Beans) d'entité pour représenter des éléments métier et une application existante qui organise les informations dans des fichiers de stockage COBOL. Par conséquent, une plateforme dont l'objectif est de simplifier la création de solutions d'intégration doit également fournir une méthode générique pour représenter des éléments métier, en faisant abstraction des techniques utilisées par les systèmes dorsaux pour la gestion des données. WebSphere Process Server et WebSphere Enterprise Service Bus permettent d'y parvenir grâce à la *structure d'objets métier*.

Cette dernière permet aux développeurs d'utiliser des schémas XML afin de définir la structure des données métier, mais aussi d'accéder et de manipuler les instances de ces structures de données (objets métier) via un code XPath ou Java. L'infrastructure d'objets métier se base sur la norme SDO (Service Data Object).

Modèle de programmation de l'architecture SCA (Service Component Architecture)

Le modèle de programmation SCA représente la base de toute solution à développer sur WebSphere Process Server et WebSphere Enterprise Service Bus. L'architecture SCA permet aux développeurs d'encapsuler les implémentations de services dans des composants réutilisables. Elle permet de définir des interfaces, des implémentations et des références indépendamment de la technologie que vous utilisez. Cette approche vous donne la possibilité d'associer des éléments à la technologie de votre choix. Il existe également un modèle de programmation client SCA qui permet d'appeler ces composants. Plus particulièrement, il permet aux infrastructures d'exécution basées sur Java d'interagir avec des exécutions non Java. L'architecture SCA utilise des objets métier comme éléments de données pour l'appel d'un service.

Outils et produits

IBM WebSphere Integration Developer est l'environnement de développement intégré qui dispose de tous les outils nécessaires pour créer et composer des

solutions d'intégration métier basées sur les technologies susmentionnées. Ces solutions sont généralement déployées dans WebSphere Process Server ou, dans certains cas de figure, dans WebSphere Enterprise Service Bus.

Chapitre 2. Liaisons

Au coeur de l'architecture orientée services se trouve le concept de *service*, une unité de fonctionnalité accomplie par une interaction entre les périphériques de traitement. Une *exportation* définit l'interface externe (ou le point d'accès) d'un module, de telle sorte que les composants SCA (Service Component Architecture) au sein du module puissent fournir leurs services à des clients externes. Une *importation* définit une interface vers des services situés à l'extérieur d'un module, de telle sorte que les services pussent être appelés depuis le module. Vous pouvez utiliser des*liaisons* spécifiques à un protocole avec des importations et des exportations pour spécifier les moyens de transport des données depuis ou vers le module.

Exportations

Les clients externes peuvent appeler des composants SCA dans un module d'intégration via une variété de protocoles (comme HTTP, JMS, MQ et RMI/IIOP) avec des données dans une variété de formats (comme XML, CSV, COBOL et JavaBean). Les exportations sont des composants qui reçoivent ces requêtes de sources externes et appellent ensuite des composants WebSphere Process Server à l'aide du modèle de programmation SCA.

Par exemple, dans la figure suivante, une exportation reçoit une requête via le protocole HTTP d'une application client. Les données sont transformées en un objet métier, le format utilisé par le composant SCA. Le composant est ensuite appelé avec cet objet de données.

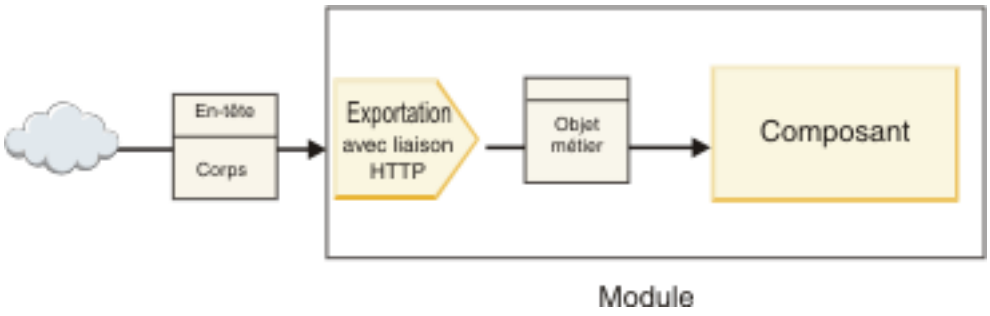


Figure 2. Exportation avec une liaison HTTP

Importations

Il est possible qu'un composant SCA souhaite appeler un service externe non SCA qui attend des données dans un format différent. Une importation est utilisée par le composant SCA pour appeler le service externe à l'aide du modèle de programmation SCA. L'importation appelle ensuite le service cible selon les attentes du service.

Par exemple, dans la figure suivante, une requête provenant d'un composant SCA est envoyée par l'importation vers un service externe. L'objet métier qui est au format utilisé par le composant SCA est transformé en format attendu par le service et le service est appelé.

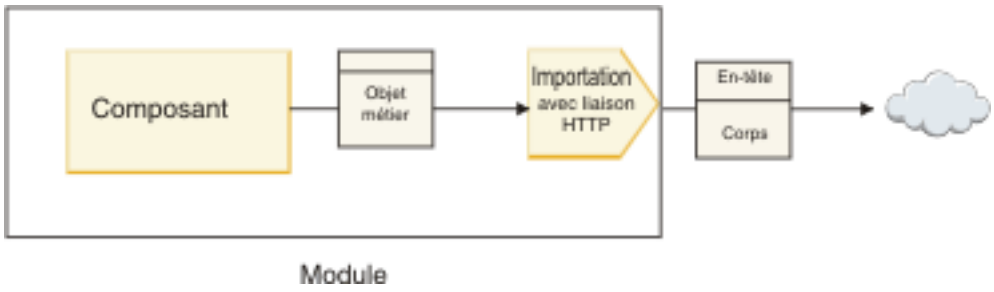


Figure 3. Importation avec une liaison HTTP

Liste de liaisons

WebSphere Integration Developer permet de générer une liaison pour une importation ou une exportation et de configurer la liaison. Les types de liaisons disponibles sont décrites dans la liste suivante.

- SCA

  La liaison SCA qui est la liaison par défaut permet à votre service de communiquer avec des services dans d'autres modules SCA. Une importation avec une liaison SCA permet d'accéder à un service dans un autre module SCA. Une exportation avec une liaison SCA permet d'offrir un service à d'autres modules SCA.
- Service Web

  Une liaison de service Web vous permet d'accéder à un service externe via des messages SOAP interopérables et des qualités de service. Vous pouvez également utiliser des liaisons de service Web pour inclure des pièces jointes dans le message SOAP.

  La liaison de service Web peut utiliser un protocole de transport de type SOAP/HTTP (SOAP via HTTP) ou SOAP/JMS (SOAP via JMS). Quel que soit le transport (HTTP ou JMS) utilisé pour transmettre les messages SOAP, les liaisons de service Web traitent toujours les interactions de demande/réponse de manière synchrone.
- HTTP

  La liaison HTTP vous permet d'accéder à un service externe via le protocole HTTP lorsque des messages non SOAP sont utilisés ou lorsque l'accès HTTP direct est requis. Cette liaison est utilisée lorsque vous travaillez avec des services Web qui sont basés sur le modèle HTTP (c'est-à-dire les services qui utilisent des opérations d'interface HTTP bien connues comme GET, PUT, DELETE, etc.).
- EJB (Enterprise JavaBeans)

  Les liaisons EJB permettent aux composants SCA d'interagir avec les services fournis par la logique métier Java EE exécutée sur un serveur Java EE.
- EIS

  Lorsqu'elle est utilisée avec un adaptateur de ressources JCA, la liaison EIS (Enterprise Information System) permet d'accéder à des services sur un système d'information d'entreprise ou de rendre vos services disponibles auprès de l'EIS.
- Liaisons JMS

  Les liaisons Java Message Service (JMS), JMS génériques et JMS WebSphere MQ (JMS MQ) sont utilisées pour des interactions avec des systèmes de messagerie où la communication asynchrone via les files d'attente de messages est critique pour la fiabilité.

Une exportation avec une des liaisons JMS surveille l'arrivée d'un message dans une file d'attente et la réponse, le cas échéant, est envoyée de manière asynchrone à la file d'attente de réponses. Une importation avec une des liaisons JMS génère et envoie un message vers une file d'attente JMS et surveille, le cas échéant, l'arrivée d'une réponse dans une file d'attente.

- JMS

  La liaison JMS vous permet d'accéder au fournisseur JMS imbriqué dans WebSphere.

- JMS générique

  La liaison JMS générique vous permet d'accéder à un système de messagerie fournisseur non IBM.

- JMS MQ

  La liaison JMS MQ vous permet d'accéder au sous-ensemble JMS d'un système de messagerie WebSphere MQ. Vous pourriez utiliser cette liaison lorsque le sous-ensemble JMS de fonctions est suffisant pour votre application.

- MQ

  La liaison WebSphere MQ vous permet de communiquer avec des applications natives MQ, en les insérant dans le cadre de l'architecture SOA et en fournissant un accès aux informations d'en-tête propres à MQ. Cette liaison est utile lorsque vous devez utiliser des fonctions natives MQ.

Concepts associés

«Présentation des liaisons d'importation et d'exportation»
Une exportation vous permet de rendre les services d'un module d'intégration disponibles pour les clients externes et une importation permet à vos composants SCA au sein d'un module d'intégration d'appeler des services externes. La liaison associée à l'exportation ou à l'importation spécifie la relation entre les messages de protocole et les objets métier. Elle spécifie également la manière dont les opérations et les erreurs sont sélectionnées.

«Configuration des liaisons d'importation et d'exportation», à la page 18
Un des aspects clés des liaisons d'importation et d'exportation est constitué par la transformation du format des données qui indique la manière dont les données sont mappées (désérialisées) à partir d'un format de connexion natif vers un objet métier ou la manière dont elles sont mappées (sérialisées) depuis un objet métier vers un format de connexion natif. Pour les liaisons associées aux exportations, vous pouvez également spécifier un sélecteur de fonction pour identifier les opérations qui doivent être effectuées sur les données. Pour les liaisons associées aux exportations ou aux importations, vous pouvez indiquer le mode de gestion des échecs qui se sont produits lors du traitement.

«Interopérabilité entre les modules SCA et les services Open SCA», à la page 32
IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture (SCA) offre un modèle de programmation simple, mais puissant, permettant de créer des applications basées sur les spécifications Open SCA. Les modules SCA de WebSphere Process Server utilisent des liaisons d'importation et d'exportation pour interagir avec les services Open SCA développés dans un environnement Rational Application Developer et hébergés par WebSphere Application Server Feature Pack for Service Component Architecture.

«Types de liaison», à la page 35
Vous pouvez utiliser des*liaisons* spécifiques à un protocole avec des importations et des exportations pour spécifier les moyens de transport des données depuis ou vers un module.

Présentation des liaisons d'importation et d'exportation

Une exportation vous permet de rendre les services d'un module d'intégration disponibles pour les clients externes et une importation permet à vos composants SCA au sein d'un module d'intégration d'appeler des services externes. La liaison associée à l'exportation ou à l'importation spécifie la relation entre les messages de protocole et les objets métier. Elle spécifie également la manière dont les opérations et les erreurs sont sélectionnées.

Flux d'informations via une exportation

Une exportation reçoit une requête destinée au composant auquel l'exportation est connectée via un transport spécifique déterminé par la liaison associée (par exemple, HTTP).

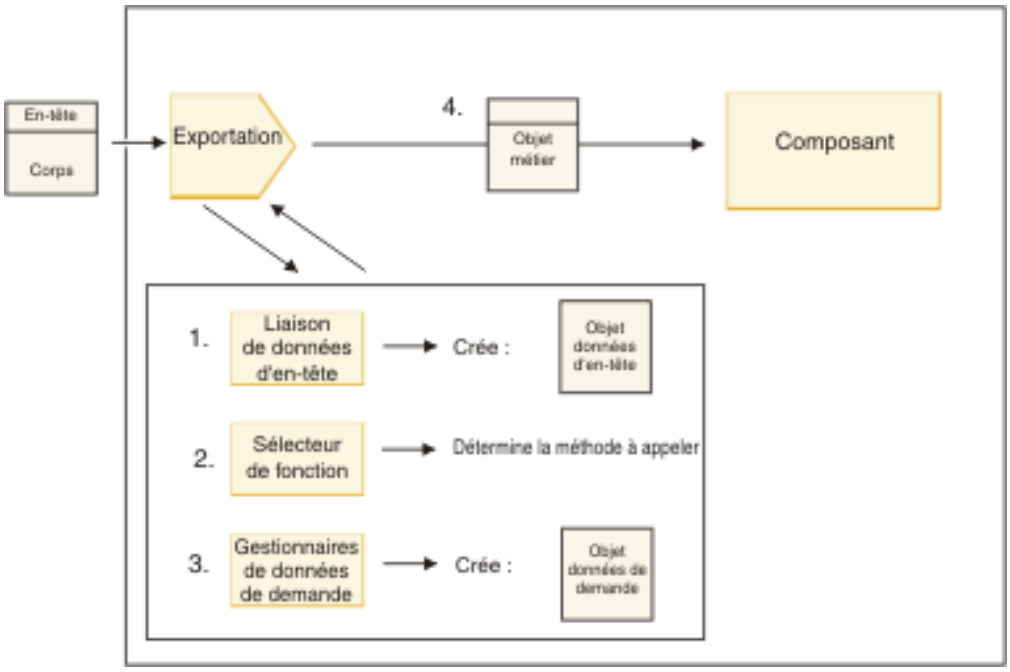


Figure 4. Flux d'une requête via l'exportation vers un composant

Lorsque l'exportation reçoit la requête, la séquence d'événements suivante se produit :

1. Pour les liaisons WebSphere MQ uniquement, la liaison des données d'en-tête transforme l'en-tête de protocole en un objet de données d'en-tête.
2. Le sélecteur de fonction détermine le nom de la méthode native à partir du message de protocole. Le nom de la méthode native est mappée par la configuration d'exportation vers le nom d'une opération sur l'interface de l'exportation.
3. Le gestionnaire de données de requête ou la liaison de données sur la méthode transforme la requête en un objet métier de type requête.
4. L'exportation appelle la méthode de composant avec l'objet métier de type requête.
   - La liaison d'exportation HTTP, la liaison d'exporation des services Web, et la liaison d'exporation des EJB appellent le composant SCA de manière synchrone.
   - Les liaisons d'exportation JMS, Generic JMS, MQ JMS et WebSphere MQ appellent le composant SCA de manière asynchrone.

Notez qu'une exportation peut propager les propriétés d'en-tête et utilisateur qu'elle reçoit via le protocole , si la propagation de contexte est activée. Les composants qui sont reliés à l'exportation peuvent ensuite accéder à ces propriétés d'en-têtes et utilisateur. Pour plus d'informations, consultez la rubrique «Propagation» du centre de documentation de WebSphere Integration Developer.

S'il s'agit d'une opération bidirectionnelle, le composant renvoie une réponse.

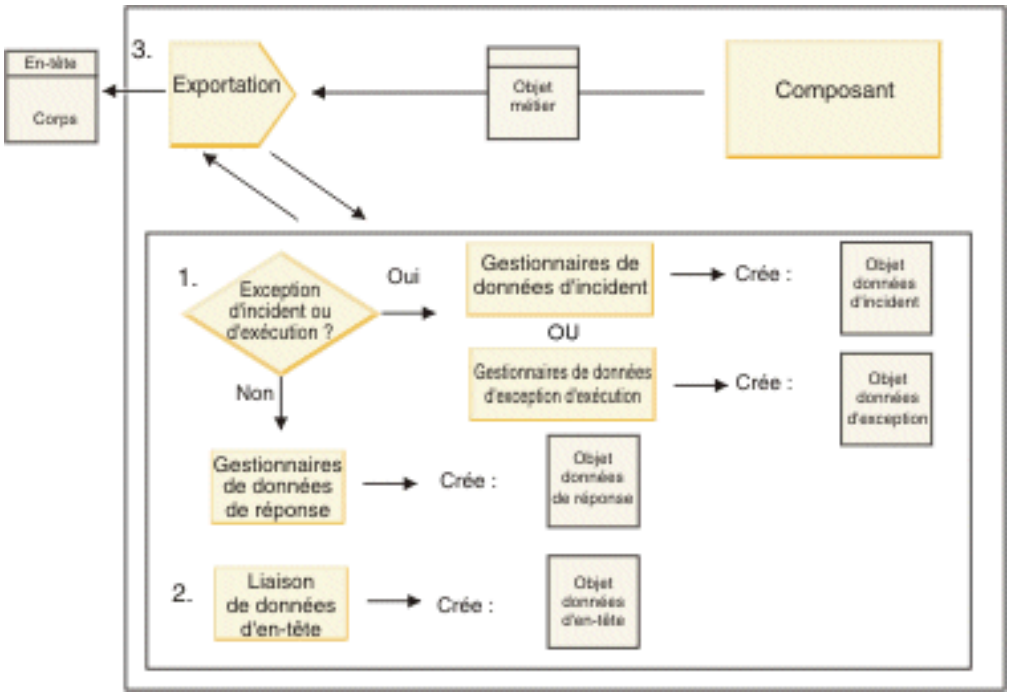


Figure 5. Flux d'une réponse en retour via l'exportation

La séquence d'étapes suivante se produit :

1. Si un message de réponse normal est reçu par la liaison d'exportation, le gestionnaire des données de réponse ou la liaison de données sur la méthode transforme l'objet métier en réponse.

   Si la réponse est une erreur, le gestionnaire des données d'erreur ou la liaison de données sur la méthode transforme l'erreur en une réponse d'erreur.

   Pour les liaisons d'exportation HTTP uniquement, si la réponse est une exception d'exécution, le gestionnaire des données d'exception d'exécution est appelé s'il est configuré.
2. Pour les liaisons WebSphere MQ uniquement, la liaison des données d'en-tête transforme les objets de données d'en-tête en en-têtes de protocole.
3. L'exportation envoie la réponse du service via le transport.

Flux d'informations via une importation

Les composants envoient des requêtes aux services en dehors du module en utilisant une importation. La requête est envoyée via un transport spécifique déterminé par la liaison associée.

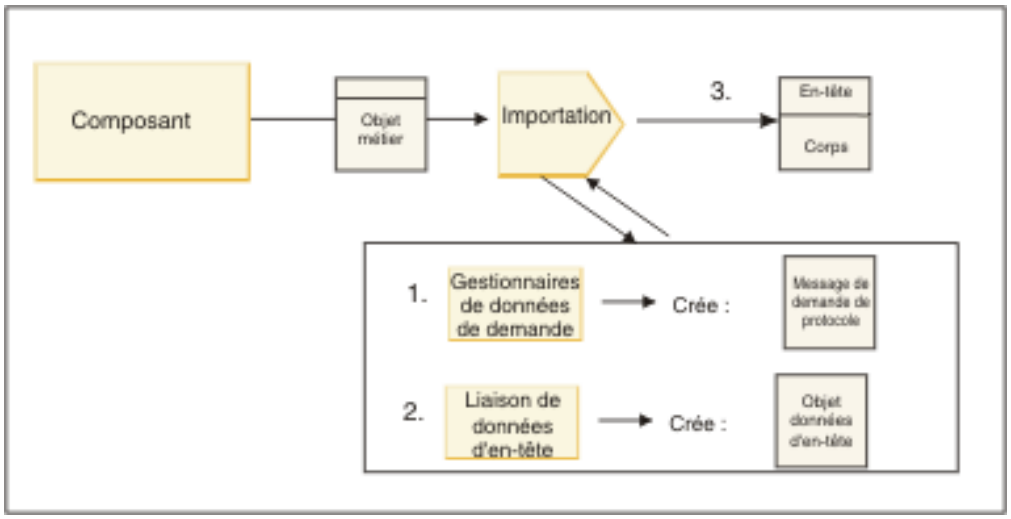


Figure 6. Flux d'un composant vers un service via l'importation

Le composant appelle l'importation avec un objet métier de type requête.

Remarque :

- Les liaisons d'importation HTTP, de services Web et EJB doivent être appelées de manière synchrone par le composant appelant.
- Les liaisons d'importation JMS, Generic JMS, MQ JMS et WebSphere MQ doivent être appelées de manière asynchrone.

Une fois que le composant a appelé l'importation, la séquence d'événements suivante se produit :

1. Le gestionnaire de données de requête ou la liaison de données sur la méthode transforme l'objet métier de type requête en un message de requête de protocole.
2. Pour les liaisons WebSphere MQ uniquement, la liaison des données d'en-tête sur la méthode définit l'objet métier d'en-tête dans l'en-tête de protocole.
3. L'importation appelle le service avec la requête de service via le transport.

S'il s'agit d'une opération bidirectionnelle, le service renvoie une réponse et la séquence d'événements suivante se produit :

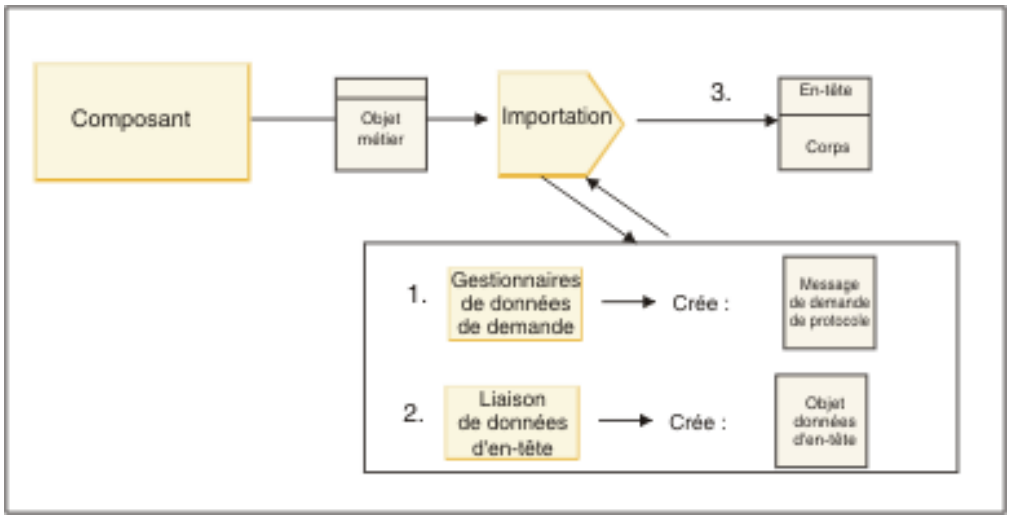


Figure 7. Flux d'une réponse en retour via l'importation

1. Pour les liaisons WebSphere MQ uniquement, la liaison des données d'en-tête transforme l'en-tête de protocole en un objet de données d'en-tête.
2. Identification de la réponse afin de déterminer s'il s'agit d'une erreur.
   - Si la réponse est une erreur, le sélecteur d'erreurs inspecte l'erreur afin de déterminer vers quelle erreur WSDL elle mappe. Le gestionnaire des données d'erreur sur la méthode transforme ensuite l'erreur en une réponse d'erreur.
   - Si la réponse est une exception d'exécution, le gestionnaire des données d'exception d'exécution est appelé s'il est configuré.
3. Le gestionnaire de données de réponse ou la liaison sur la méthode transforme la réponse en un objet métier de réponse.
4. L'importation renvoie l'objet métier de réponse au composant.

Configuration des liaisons d'importation et d'exportation

Un des aspects clés des liaisons d'importation et d'exportation est constitué par la transformation du format des données qui indique la manière dont les données sont mappées (désérialisées) à partir d'un format de connexion natif vers un objet métier ou la manière dont elles sont mappées (sérialisées) depuis un objet métier vers un format de connexion natif. Pour les liaisons associées aux exportations, vous pouvez également spécifier un sélecteur de fonction pour identifier les opérations qui doivent être effectuées sur les données. Pour les liaisons associées aux exportations ou aux importations, vous pouvez indiquer le mode de gestion des échecs qui se sont produits lors du traitement.

En outre, vous pouvez spécifier des informations spécifiques au transport sur les liaisons. Par exemple, pour une liaison HTTP, vous pouvez spécifier l'adresse URL de noeud final. Pour plus d'informations, voir le centre de documentation de WebSphere Integration Developer. Par exemple, pour la liaison HTTP, les informations spécifiques au transport sont décrites dans les rubriques «Génération d'une liaison d'importation HTTP» et «Génération d'une liaison d'exportation HTTP».

Concepts associés

«Transformation du format des données dans les importations et exportations»
Lorsqu'une liaison d'exportation ou d'importation est configurée dans WebSphere Integration Developer, une des propriétés de configuration spécifiées concerne le format de données utilisé par la liaison.

«Sélecteurs de fonction dans les liaisons d'exportation», à la page 24
Un sélecteur de fonction permet d'identifier les opérations qui doivent être effectuées sur les données pour un message de demande. Les sélecteurs de fonction sont configurés dans le cadre d'une liaison d'exportation.

«Gestion d'erreurs», à la page 26
Vous pouvez configurer vos liaisons d'importation et d'exportation pour gérer les erreurs (par exemple, les exceptions métier) qui se produisent lors du traitement en spécifiant des gestionnaires de données d'erreur. Vous pouvez configurer un gestionnaire de données d'erreur sur trois niveaux – vous pouvez associer un gestionnaire de données d'erreur à une erreur, à une opération ou pour toutes les opérations à une liaison.

Transformation du format des données dans les importations et exportations

Lorsqu'une liaison d'exportation ou d'importation est configurée dans WebSphere Integration Developer, une des propriétés de configuration spécifiées concerne le format de données utilisé par la liaison.

- Pour les liaisons d'exportation, dans lesquelles une application client envoie des requêtes à un composant SCA et reçoit des réponses en retour, vous indiquez le format des données natives. En fonction du format, le système sélectionne le gestionnaire de données approprié ou la liaison de données pour transformer les données natives en un objet métier (qui est utilisé par le composant SCA) et inversement pour transformer l'objet métier en données natives (qui est la réponse à l'application client).
- Pour les liaisons d'importation, dans lesquelles un composant SCA envoie des requêtes à un service en dehors du module et reçoit des réponses en retour, vous indiquez le format de données des données natives. En fonction du format, le système sélectionne le gestionnaire de données approprié ou la liaison de données pour transformer l'objet métier en données natives et inversement.

WebSphere Process Server fournit un ensemble de formats de données prédéfinis et de gestionnaires de données correspondants ou de liaisons de données qui prennent en charge les formats. Vous pouvez également créer vos propres gestionnaires de données personnalisés et enregistrer le format de données pour ces gestionnaires de données. Pour plus d'informations, consultez la rubrique «Développement de gestinnaires de données» du centre de documentation de WebSphere Integration Developer.

- Les *gestionnaires de données* ne dépendent pas d'un protocole et peuvent transformer les données d'un format à un autre. DansWebSphere Process Server, les gestionnaires de données transforment généralement des données natives (comme XML, CSV et COBOL) en un objet métier et inversement. Puisqu'ils ne dépendent pas d'un protocole, vous pouvez réutiliser le même gestionnaire de données avec toute une variété de liaisons d'exportation et d'importation. Par exemple, vous pouvez utiliser le même gestionnaire de données XML avec une liaison d'exportation ou d'importation HTTP ou avec une liaison d'exportation ou d'importation JMS.

- Les *liaisons de données* transforment également des données natives en objet métier (et inversement), mais elles sont spécifiques à un protocole. Par exemple, une liaison de données HTTP peut être utilisée uniquement avec une liaison d'exportation ou d'importation HTTP. Contrairement aux gestionnaires de données, une liaison de données HTTP ne peut pas être réutilisée avec une liaison d'exportation ou d'importation MQ.

  Remarque : Trois liaisons de données HTTP (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML et HTTPServiceGatewayDataBinding) sont dépréciées à partir de WebSphere Process Server, version 7.0. Utilisez des gestionnaires de données chaque fois que possible.

Comme indiqué précédemment, vous pouvez créer des gestionnaires de données personnalisés, si nécessaire. Vous pouvez également créer des liaisons de données personnalisées. Toutefois, il est recommandé de créer des gestionnaires de données personnalisés car ils peuvent être utilisés avec plusieurs liaisons.

Concepts associés

«Gestionnaires de données»
Les gestionnaires de données sont configurés en fonction des liaisons d'importation et d'exportation pour transformer les données d'un format à un autre dans un environnement de protocole neutre. Plusieurs gestionnaires de données sont fournis avec le produit, mais vous pouvez également créer votre propre gestionnaire de données si nécessaire. Vous pouvez associer un gestionnaire de données à une liaison d'importation ou d'exportation à l'un des deux niveaux disponibles : vous pouvez l'associer à toutes les opérations dans l'interface de l'importation ou de l'exportation ou bien vous pouvez l'associer à une opération spécifique pour la requête ou la réponse.

«Liaisons de données», à la page 22
Les liaisons de données sont configurées en fonction des liaisons d'importation et d'exportation pour transformer les données d'un format à un autre. Les liaisons de données sont spécifiques à un protocole. Plusieurs liaisons de données sont fournies avec le produit, mais vous pouvez également créer votre propre liaison de données si nécessaire. Vous pouvez associer une liaison de données à une liaison d'importation ou d'exportation sur l'un des deux niveaux disponibles – vous pouvez l'associer avec toutes les opérations dans l'interface de l'importation ou de l'exportation ou bien vous pouvez l'associer à une opération spécifique pour la requête ou la réponse.

Gestionnaires de données

Les gestionnaires de données sont configurés en fonction des liaisons d'importation et d'exportation pour transformer les données d'un format à un autre dans un environnement de protocole neutre. Plusieurs gestionnaires de données sont fournis avec le produit, mais vous pouvez également créer votre propre gestionnaire de données si nécessaire. Vous pouvez associer un gestionnaire de données à une liaison d'importation ou d'exportation à l'un des deux niveaux disponibles : vous pouvez l'associer à toutes les opérations dans l'interface de l'importation ou de l'exportation ou bien vous pouvez l'associer à une opération spécifique pour la requête ou la réponse.

Gestionnaires de données prédéfinis

WebSphere Integration Developer permet de spécifier le gestionnaire de données que vous voulez utiliser.

Le tableau suivant répertorie les gestionnaires de données prédéfinis et décrit également la manière dont chaque gestionnaire de données transforme les données entrantes et sortantes.

Remarque : Sauf indication contraire, ces gestionnaires de données peuvent être utilisés avec des liaisons JMS, Generic JMS, JMS MQ, WebSphere MQ et HTTP. Voir la rubrique «Gestionnaires de données» dans le centre de documentation de WebSphere Integration Developer pour obtenir des informations détaillées.

Tableau 1. Gestionnaires de données prédéfinis

Gestionnaire de données	Données natives vers objet métier	Objet métier vers données natives
ATOM	Analyse les flux ATOM dans un objet métier de flux ATOM.	Sérialise un objet métier de flux ATOM vers des flux ATOM.
Délimité	Analyse les données délimitées dans un objet métier.	Sérialise un objet métier vers des données délimitées, y compris CSV.
Largeur fixe	Analyse les données à largeur fixe dans un objet métier.	Sérialise un objet métier vers des données à largeur fixe.
Géré par WTX	Délègue la transformation du format des données à WebSphere Transformation Extender (WTX). Le nom de mappe WTX est dérivé par le gestionnaire de données.	Délègue la transformation du format des données à WebSphere Transformation Extender (WTX). Le nom de mappe WTX est dérivé par le gestionnaire de données.
Géré par le générateur d'appels WTX	Délègue la transformation du format des données à WebSphere Transformation Extender (WTX). Le nom de mappe WTX est fourni par l'utilisateur.	Délègue la transformation du format des données à WebSphere Transformation Extender (WTX). Le nom de mappe WTX est fourni par l'utilisateur.
JAXB	Sérialise des beans Java sous la forme d'objet métier à l'aide de règles de mappage définies par la spécification JAXB (Java Architecture for XML Binding).	Désérialise un objet métier Java à l'aide des règles de mappage définies par la spécification JAXB.
JAXWS **Remarque :** Le gestionnaire de données JAXWS ne peut être utilisé qu'avec la liaison EJB.	Utilisé par une liaison d'EJB pour transformer un objet Java de réponse ou un objet Java d'exception en objet métier de réponse à l'aide des règles de mappage définies par la spécification JAX-WS (Java API for XML Web Services).	Utilisé par une liaison d'EJB pour transformer un objet métier en paramètre de méthode Java à l'aide des règles de mappage définies par la spécification JAX-WS.
JSON	Analyse les données JSON dans un objet métier.	Sérialise un objet métier vers des données JSON.
Corps natif	Effectue l'analyse syntaxique des octets natifs (texte, mappe, flux ou objet) pour obtenir un des cinq objets métier de base (texte, octets, mappe, flux ou objet).	Transforme les cinq objets métier de base en octets, texte, mappe, flux ou objet.

Tableau 1. Gestionnaires de données prédéfinis (suite)

Gestionnaire de données	Données natives vers objet métier	Objet métier vers données natives
SOAP	Analyse le message SOAP (et l'en-tête) dans un objet métier.	Sérialise un objet métier vers un message SOAP.
XML	Analyse les données XML dans un objet métier.	Sérialise un objet métier vers des données XML.
UTF8XMLDataHandler	Analyse les données XML codées en UTF-8 dans un objet métier.	Sérialise un objet métier dans des données XML codées en UTF-8 lors de l'envoi d'un message.

Création d'un gestionnaire de données

Pour plus d'informations sur la création d'un gestionnaire de données, consultez la rubrique «Développement des gestionnaires de données» du centre de documentation deWebSphere Integration Developer.

Liaisons de données

Les liaisons de données sont configurées en fonction des liaisons d'importation et d'exportation pour transformer les données d'un format à un autre. Les liaisons de données sont spécifiques à un protocole. Plusieurs liaisons de données sont fournies avec le produit, mais vous pouvez également créer votre propre liaison de données si nécessaire. Vous pouvez associer une liaison de données à une liaison d'importation ou d'exportation sur l'un des deux niveaux disponibles – vous pouvez l'associer avec toutes les opérations dans l'interface de l'importation ou de l'exportation ou bien vous pouvez l'associer à une opération spécifique pour la requête ou la réponse.

WebSphere Integration Developer permet de spécifier quelle liaison de données vous voulez utiliser ou de créer votre propre liaison de données. Une discussion traitant des liaisons de données est disponible dans la section «Présentation des liaisons JMS, JMS MQ et JMS génériques» du centre de documentation de WebSphere Integration Developer.

Liaisons JMS

Le tableau suivant répertorie les liaisons de données qui peuvent être utilisées :

- Liaisons JMS
- Liaisons JMS génériques
- Liaisons JMS WebSphere MQ

Le tableau comprend également une description des tâches que les liaisons de données effectuent.

Tableau 2. Liaisons de données prédéfinies pour les liaisons JMS

Liaison de données	Données natives vers objet métier	Objet métier vers données natives
Objet Java sérialisé	Transforme l'objet sérialisé Java en un objet métier (qui est mappé en tant que type d'entrée ou de sortie dans WSDL).	Sérialise un objet métier vers un objet sérialisé Java dans le message d'objet JMS.
Octets encapsulés	Extrait les octets du message d'octets JMS entrant et les encapsule dans l'objet métier JMSBytesBody.	Extrait les octets de l'objet métier JMSBytesBody et les encapsule dans le message d'octets JMS sortant
Entrée de mappe encapsulée	Extrait les informations de nom, de valeur et de type pour chaque entrée dans le message de mappe JMS entrant et crée une liste d'objets métier MapEntry. Encapsule ensuite la liste dans l'objet métier JMSMapBody	Extrait les informations de nom, de valeur et de type à partir de la liste MapEntry dans l'objet métier JMSMapBody business et crée les entrées correspondantes dans le message de mappe JMS sortant.
Objet encapsulé	Extrait l'objet du message d'objet JMS entrant et l'encapsule dans l'objet métier JMSObjectBody.	Extrait l'objet de l'objet métier JMSObjectBody et l'encapsule dans le message d'objet JMS sortant.
Texte encapsulé	Extrait le texte du message de texte JMS entrant et l'encapsule dans l'objet métier JMSTextBody.	Extrait le texte de l'objet métier JMSTextBody et l'encapsule dans le message de texte JMS sortant.

Liaisons WebSphere MQ

Le tableau suivant répertorie les liaisons de données qui peuvent être utilisées avec WebSphere MQ et décrit les tâches que les liaisons de données peuvent effectuer.

Tableau 3. Liaisons de données prédéfinies pour les liaisons WebSphere MQ

Liaison de données	Données natives vers objet métier	Objet métier vers données natives
Objet Java sérialisé	Transforme l'objet sérialisé Java à partir du mesage entrant en un objet métier (qui est mappé en tant que type d'entrée ou de sortie dans WSDL).	Transforme un objet métier en objet sérialisé Java dans le message sortant
Octets encapsulés	Extrait les octets du message d'octets MQ non structuré et les encapsule dans l'objet métier JMSBytesBody.	Extrait les octets d'un objet métier JMSBytesBody et encapsule ces octets dans le message d'octets MQ non structuré sortant.
Texte encapsulé	Extrait le texte du message de texte MQ non structuré et l'encapsule dans un objet métier JMSTextBody.	Extrait le texte d'un objet métier JMSTextBody et l'encapsule dans un message de texte MQ non structuré.

Tableau 3. Liaisons de données prédéfinies pour les liaisons WebSphere MQ (suite)

Liaison de données	Données natives vers objet métier	Objet métier vers données natives
Entrée de flux encapsulée	Extrait les informations de nom et de type pour chaque entrée dans le message de flux JMS entrant et crée une liste d'objets métier StreamEntry. Encapsule ensuite la liste dans l'objet métierJMSStreamBody.	Extrait les informations de nom et de type à partir de la liste StreamEntry dans l'objet métier JMSStreamBody et crée les entrées correspondantes dans le message JMSStreamMessage sortant.

Outre les liaisons de données répertoriées dans tableau 3, à la page 23, WebSphere MQ utilise également des liaisons de données d'en-tête. Voir le centre de documentation de WebSphere Integration Developer pour des informations détaillées.

Liaisons HTTP

Le tableau suivant répertorie les liaisons de données qui peuvent être utilisées avec HTTP et décrit les tâches que les liaisons de données peuvent effectuer.

Tableau 4. Liaisons de données prédéfinies pour les liaisons HTTP

Liaison de données	Données natives vers objet métier	Objet métier vers données natives
Octets encapsulés	Extrait les octets du corps du message HTTP entrant et les encapsule dans l'objet métier HTTPBytes.	Extrait les octets de l'objet métier HTTPBytes et les ajoute au corps du message HTTP sortant.
Texte encapsulé	Extrait le texte du corps du message HTTP entrant et l'encapsule dans l'objet métier HTTPText.	Extrait le texte de l'objet métier HTTPText et l'ajoute au corps du message HTTP sortant.

Sélecteurs de fonction dans les liaisons d'exportation

Un sélecteur de fonction permet d'identifier les opérations qui doivent être effectuées sur les données pour un message de demande. Les sélecteurs de fonction sont configurés dans le cadre d'une liaison d'exportation.

Considérons une exportation SCA qui expose une interface. L'interface contient deux opérations – Créer et Mettre à jour. L'exportation a une liaison JMS qui lit à partir d'une file d'attente.

Lorsqu'un message arrive dans la file d'attente, l'exportation est transmise aux données associées, mais quelle opération de l'interface d'exportation doit être appelée sur le composant connecté ? L'opération est déterminée par le sélecteur de fonction et la configuration de la liaison d'exportation.

Le sélecteur de fonction renvoie le nom de la fonction native (le nom de la fonction dans le système client ayant envoyé le message). Le nom de la fonction native est ensuite mappé à l'opération ou au nom de la fonction sur l'interface associée à l'exportation. Par exemple, dans la figure suivante, le sélecteur de fonction renvoie le nom de la fonction native (CRT) à partir du message entrant, le nom de la fonction native est mappé à l'opération Créer et l'objet métier est envoyé au

composant SCA avec l'opération Créer.

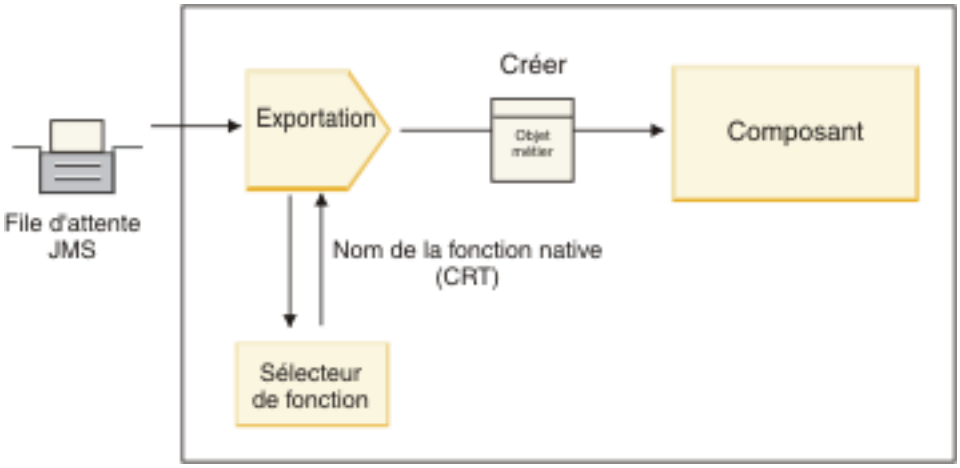


Figure 8. Le sélecteur de fonction

Si l'interface ne présente qu'une opération, il n'est pas nécessaire de spécifier un sélecteur de fonction.

Plu sieurs sélecteurs de fonction préintégrés sont disponibles et répertoriés dans les sections ci-dessous.

Liaisons JMS

Le tableau suivant répertorie les sélecteurs de fonction qui peuvent être utilisés avec :

- Liaisons JMS
- Liaisons JMS génériques
- Liaisons JMS WebSphere MQ

Tableau 5. Sélecteurs de fonction prédéfinis pour les liaisons JMS

Sélecteur de fonction	Description
Sélecteur de fonction JMS pour des liaisons de données JMS simples	Utilise la propriété JMSType du message pour sélectionner le nom de l'opération.
Sélecteur de fonction de propriété d'en-tête JMS	Renvoie la valeur de la propriété de chaîne JMS, TargetFunctionName, à partir de l'en-tête.
Sélecteur de fonction de passerelle de service JMS	Détermine si la requête est une opération unidirectionnelle ou bidirectionnelle en examinant l'ensemble de propriétés JMSReplyTo par le client.

Liaisons WebSphere MQ

Le tableau suivant répertorie les sélecteurs de fonction qui peuvent être utilisés avec les liaisonsWebSphere MQ.

Tableau 6. Sélecteurs de fonction prédéfinis pour les liaisons WebSphere MQ

Sélecteur de fonction	Description
Sélecteur de fonction MQ handleMessage	Renvoie un message handleMessage comme valeur qui est mappé à l'aide des liaisons de méthode d'exportation au nom d'une opération sur l'interface.
MQ utilise le sélecteur de fonction JMS par défaut	Lit l'opération native à partir de la propriété TargetFunctionName du dossier d'un en-tête MQRFH2.
MQ utilise le format du corps de message comme fonction native	Recherche la zone Format du dernier en-tête et renvoie cette zone sous forme de chaîne.
Sélecteur de fonction de type MQ	Crée une méthode dans votre liaison d'exportation par extraction de l'adresse URL contenant les propriétés Msd, Set, Type et Format de l'en-tête MQRFH2.
Sélecteur de fonction de passerelle de service MQ	Utilise la propriété MsgType de l'en-tête MQMD pour déterminer le nom de l'opération.

Liaisons HTTP

Le tableau suivant répertorie les sélecteurs de fonction qui peuvent être utilisés avec les liaisons HTTP.

Tableau 7. Sélecteurs de fonction prédéfinis pour les liaisons HTTP

Sélecteur de fonction	Description
Sélecteur de fonction HTTP basé sur l'en-tête TargetFunctionName	Utilise la propriété d'en-tête HTTP TargetFunctionName HTTP provenant du client pour déterminer l'opération à appeler en phase d'exécution à partir de l'exportation.
Sélecteur de fonction HTTP basé sur l'adresse URL et la méthode HTTP	Utilise le chemin relatif à partir de l'adresse URL ajoutée avec la méthode HTTP à partir du client pour déterminer l'opération native définie sur l'exporation.
Sélecteur de fonction de passerelle de service HTTP basé sur l'adresse URL avec un nom d'opération	Détermine la méthode à appeler en fonction de l'adresse URL si "operationMode = oneWay" a été ajouté à l'URL de demande.

Remarque : Vous pouvez également créer votre propre sélecteur de fonction à l'aide de WebSphere Integration Developer. Pour plus d'informations sur la création d'un sélecteur de fonction, consultez le centre de documentation de WebSphere Integration Developer. Par exemple, une description de la création d'un sélecteur de fonction pour les liaisons WebSphere MQ est disponible dans la rubrique «Présentation des sélecteurs de fonction MQ».

Gestion d'erreurs

Vous pouvez configurer vos liaisons d'importation et d'exportation pour gérer les erreurs (par exemple, les exceptions métier) qui se produisent lors du traitement en spécifiant des gestionnaires de données d'erreur. Vous pouvez configurer un

gestionnaire de données d'erreur sur trois niveaux – vous pouvez associer un gestionnaire de données d'erreur à une erreur, à une opération ou pour toutes les opérations à une liaison.

Un gestionnaire de données d'erreur traite les données d'erreur et les convertit au format correct pour les envoyer via la liaison d'importation ou d'exportation.

- Pour une liaison d'exportation, le gestionnaire de données d'erreur transforme l'objet métier d'exception envoyé depuis le composant en un message de réponse qui peut être utilisé par l'application client.
- Pour une liaison d'importation, le gestionnaire de données d'erreur transforme les données d'erreur ou le message de réponse provenant d'un service en un objet métier d'exception qui peut être utilisé par le composant SCA.

Pour les liaisons d'importation, la liaison appelle le sélecteur d'erreurs qui détermine si le message de réponse est une réponse normale, une erreur métier ou une exception d'exécution.

Vous pouvez spécifier un gestionnaire de données d'erreur pour une erreur, une opération particulière et pour toutes les opérations avec une liaison.

- Si le gestionnaire de données d'erreur est défini sur les trois niveaux, le gestionnaire de données associé à une erreur particulière est appelé.
- Si les gestionnaires de données d'erreur sont définis aux niveaux de l'opération et de la liaison, le gestionnaire de données associé à l'opération est appelé.

Deux éditeurs sont utilisés dans WebSphere Integration Developer pour spécifier la gestion des erreurs. L'éditeur d'interface est utilisé pour indiquer s'il y aura une erreur sur une opération. Après avoir généré une liaison avec cette interface, l'éditeur de la vue des propriétés vous permet de configurer la manière dont l'erreur sera gérée. Pour plus d'informations, consultez la rubrique «Sélecteurs d'ereurs» du centre de documentation de WebSphere Integration Developer.

Concepts associés

«Gestion des erreurs dans les liaisons d'exportation»
Lorsqu'une erreur se produit lors du traitement de la requête d'une application client, la liaison d'exportation peut renvoyer les informations d'erreur au client. Vous pouvez configurer la liaison d'exportation pour spécifier la manière dont l'erreur doit être traitée et renvoyée au client.

«Gestion des erreurs dans les liaisons d'importation», à la page 29
Un composant utilise une importation pour envoyer une requête à un service en dehors du module. Lorsqu'une erreur se produit lors du traitement de la requête, le service renvoie l'erreur à la liaison d'importation. Vous pouvez configurer la liaison d'importation pour spécifier la manière dont l'erreur doit être traitée et renvoyée au composant.

Gestion des erreurs dans les liaisons d'exportation

Lorsqu'une erreur se produit lors du traitement de la requête d'une application client, la liaison d'exportation peut renvoyer les informations d'erreur au client. Vous pouvez configurer la liaison d'exportation pour spécifier la manière dont l'erreur doit être traitée et renvoyée au client.

Vous pouvez configurer la liaison d'exportation à l'aide de WebSphere Integration Developer.

Lors du traitement de la requête, un client appelle une exportation avec une requête et l'exportation appelle le composant SCA. Lors du traitement de la

requête, le composant SCA peut renvoyer une réponse métier ou émettre une exception métier de service ou une exception d'exécution de service. Lorsque cela se produit, la liaison d'exportation transforme l'exception en un message d'erreur et l'envoie au client, comme indiqué dans la figure ci-dessous et décrit dans les sections suivantes.

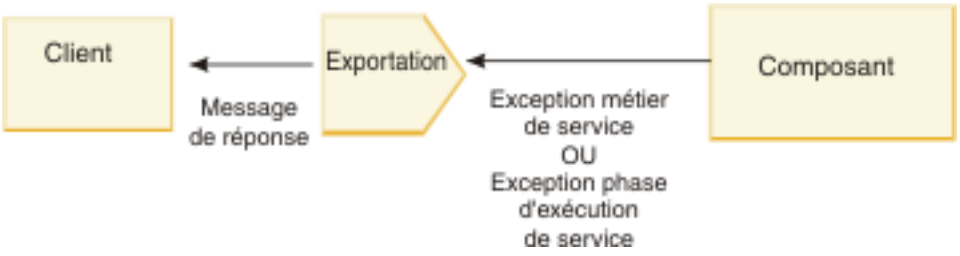


Figure 9. Manière dont les informations sont envoyées du composant vers le client via la liaison d'exportation

Vous pouvez créer un gestionnaire de données personnalisé ou une liaison de données pour gérer les erreurs.

Erreurs métier

Les erreurs métier sont des erreurs ou des exceptions métier qui se produisent lors du traitement.

Considérez l'interface suivante présentant une opération createCustomer. Cette opération se caractérise par deux erreurs métier définies : CustomerAlreadyExists et MissingCustomerId.

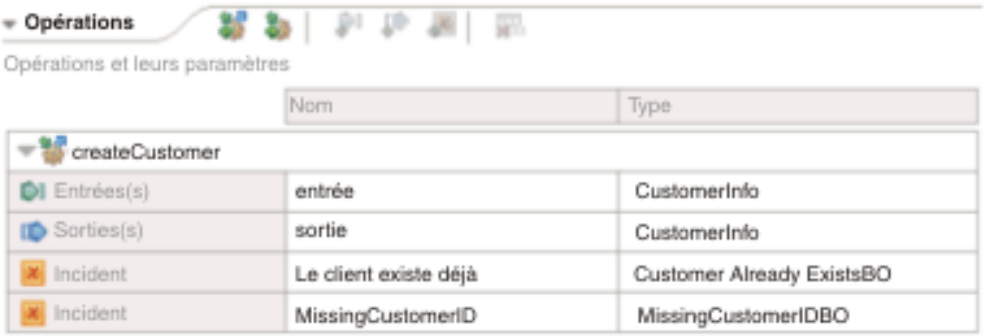


Figure 10. Interface avec deux erreurs

Dans cet exemple, si un client envoie une requête pour créer un client (à ce composant SCA) et que le client existe déjà, le composant émet une erreur CustomerAlreadyExists vers l'exportation. L'exportation doit propager cette erreur métier en retour au client appelant. Pour ce faire, elle utilise le gestionnaire de données configuré sur la liaison d'exportation.

Lorsqu'une erreur métier est reçue par la liaison d'exportation, le traitement suivant se produit :

1. La liaison détermine le gestionnaire de données d'erreurs à appeler pour gérer les erreurs. Si l'exception métier de service contient le nom de l'erreur, le

gestionnaire de données configuré sur l'erreur est appelé. Si l'exception métier de service ne contient pas le nom de l'erreur, ce nom est dérivé en faisant correspondre les types d'erreur.

2. La liaison appelle le gestionnaire de données d'erreurs avec l'objet de données à partir de l'exception métier de service.
3. Le gestionnaire de données d'erreur transforme l'objet données d'erreur en un message de réponse et le renvoie à la liaison d'exportation.
4. L'exportation renvoie le message de réponse au client.

Si l'exception métier de service contient le nom de l'erreur, le gestionnaire de données configuré sur l'erreur est appelé. Si l'exception métier de service ne contient pas le nom de l'erreur, ce nom est dérivé en faisant correspondre les types d'erreur.

Exceptions d'exécution

Une exception d'exécution est une exception qui se produit dans l'application SCA lors du traitement d'une requête qui ne correspond pas à une erreur métier. Contrairement aux erreurs métier, les exceptions d'exécution ne sont pas définies sur l'interface.

Dans certains scénarios, il est possible que vous souhaitiez propager ces exceptions d'exécution à l'application client de telle sorte que l'application client puisse entreprendre l'action appropriée.

Par exemple, si un client envoie une requête (au composant SCA) pour créer un client et qu'une erreur d'autorisation se produit lors du traitement de la requête, le composant émet une exception d'exécution. Cette exception d'exécution doit être propagée en retour au client appelant de telle sorte qu'il puisse entreprendre l'action appropriée en ce qui concerne l'autorisation. Cette opération est accomplie par le gestionnaire de données d'exception d'exécution configuré sur la liaison d'exportation.

Remarque : Vous pouvez configurer un gestionnaire de données d'exception d'exécution uniquement sur des liaisons HTTP.

Le traitement d'une exception d'exécution est identique au traitement d'une erreur métier. Si un gestionnaire de données d'exception d'exécution était configuré, le traitement suivant se produit :

1. La liaison d'exportation appelle le gestionnaire de données approprié avec l'exception d'exécution de service.
2. Le gestionnaire de données transforme l'objet données d'erreur en un message de réponse et le renvoie à la liaison d'exportation.
3. L'exportation renvoie le message de réponse au client.

La gestion des erreurs et la gestion des exceptions d'exécution sont facultatives. Si vous ne voulez pas propager les erreurs et les exceptions d'exécution au client appelant, ne configurez pas le gestionnaire de données d'erreur ou le gestionnaire de données d'exception d'exécution.

Gestion des erreurs dans les liaisons d'importation

Un composant utilise une importation pour envoyer une requête à un service en dehors du module. Lorsqu'une erreur se produit lors du traitement de la requête,

le service renvoie l'erreur à la liaison d'importation. Vous pouvez configurer la liaison d'importation pour spécifier la manière dont l'erreur doit être traitée et renvoyée au composant.

Vous pouvez configurer la liaison d'importation à l'aide de WebSphere Integration Developer. Vous pouvez spécifier un gestionnaire de données d'erreur (ou une liaison de données) et vous pouvez également spécifier un sélecteur d'erreurs.

Gestionnaires de données d'erreur

Le service qui traite la requête envoie des informations d'erreur à la liaison d'importation sous la forme d'une exception ou un message de réponse contenant les données d'erreur.

La liaison d'importation transforme l'exception de service ou le message de réponse en une exception métier de service ou une exception d'exécution de service, comme indiqué dans la figure ci-dessous et décrit dans les sections suivantes.

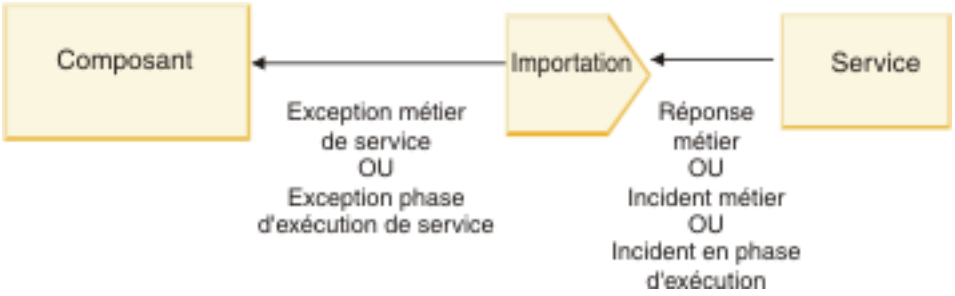


Figure 11. Manière dont les informations d'erreur sont envoyées depuis le service au composant via l'importation

Vous pouvez créer un gestionnaire de données personnalisé ou une liaison de données pour gérer les erreurs.

Sélecteurs d'erreurs

Lorsque vous configurez une liaison d'importation, vous pouvez spécifier un sélecteur d'erreurs. Le sélecteur d'erreurs détermine si la réponse d'importation est une réponse réelle, une exception métier ou une erreur d'exécution. Il détermine également, à partir de l'en-tête ou du corps de la réponse, le nom de l'erreur native qui est mappé par la configuration de liaison au nom d'une erreur dans l'interface associée.

Deux types de sélecteurs d'erreurs préintégrés sont disponibles avec les importations JMS, JMS MQ, JMS génériques, WebSphere MQ et HTTP :

Tableau 8. Sélecteurs d'erreurs préintégrés

Type de sélecteur d'erreurs	Description
Basé sur les en-têtes	Indique si un message de réponse est une erreur métier, une exception d'exécution ou un message normal en fonction des en-têtes dans le message de réponse.

Tableau 8. Sélecteurs d'erreurs préintégrés (suite)

Type de sélecteur d'erreurs	Description
SOAP	Détermine si le message de réponse SOAP est une réponse normale, une erreur métier ou une exception d'exécution.

Les exemples ci-dessous illustrent des sélecteurs d'erreurs basés sur les en-têtes et le sélecteur d'erreurs SOAP.

- Sélecteur d'erreurs basé sur les en-têtes

  Si une application veut indiquer que le message entrant est une erreur métier, il doit y avoir deux en-têtes dans le message entrant pour les erreurs métier, comme indiqué ci-après :

  ```
  Header name = FaultType, Header value = Business
  Header name = FaultName, Header value = <user defined native fault name>
  ```

  Si une application veut indiquer que le message de réponse entrant est une exception d'exécution, il doit y avoir un en-tête dans le message entrant comme indiqué ci-après :

  ```
  Header name = FaultType, Header value = Runtime
  ```

- Sélecteur d'erreurs SOAP

  Une erreur métier peut être envoyée dans le cadre d'un message SOAP avec l'en-tête SOAP personnalisé suivant : "CustomerAlreadyExists" est le nom de l'erreur dans ce cas.

  ```
  <ibmSoap:BusinessFaultName
  xmlns:ibmSoap="http://www.ibm.com/soap">CustomerAlreadyExists
  <ibmSoap:BusinessFaultName>
  ```

Le sélecteur d'erreurs est facultatif. Si vous ne spécifiez aucun sélecteur d'erreurs, la liaison d'importation ne peut pas déterminer le type de réponse. En conséquence, la liaison la traite comme une réponse métier et appelle le gestionnaire de données de réponse ou la liaison de données.

Vous pouvez créer un sélecteur d'erreurs personnalisé. Les étapes permettant de créer un sélecteur d'erreurs personnalisé sont indiquées dans la rubrique «Développement d'un sélecteur d'erreurs personnalisé» du centre de documentation de WebSphere Integration Developer.

Erreurs métier

Une erreur métier peut se produire lorsqu'il existe une erreur dans le traitement d'une requête. Par exemple, si vous envoyez une requête pour créer un client et que le client existe déjà, le service envoie une exception métier à la liaison d'importation.

Lorsqu'une exception métier est reçue par la liaison, les étapes du traitement dépendent de la configuration d'un sélecteur d'erreurs pour la liaison.

- Si aucun sélecteur d'erreurs n'a été configuré, la liaison appelle le gestionnaire de données de réponse ou la liaison de données.
- Si un sélecteur d'erreurs a été configuré, le traitement suivant se produit :
  1. La liaison d'importation appelle le sélecteur d'erreurs pour déterminer si la réponse est une erreur métier, une réponse métier ou une erreur d'exécution.
  2. Si la réponse est une erreur métier, la liaison d'importation appelle le sélecteur d'erreurs pour fournir le nom de l'erreur native.

3. La liaison d'importation identifie l'erreur WSDL correspondant au nom de l'erreur native renvoyée par le sélecteur d'erreurs.
4. La liaison d'importation détermine le gestionnaire de données d'erreur configuré pour cette erreur WSDL.
5. La liaison d'importation appelle ce gestionnaire de données d'erreur avec les données d'erreur.
6. Le gestionnaire de données d'erreur transforme les données d'erreur en un objet de données et le renvoie à la liaison d'importation.
7. La liaison d'importation construit un objet d'exception métier de service avec l'objet de données et le nom de l'erreur.
8. L'importation renvoie l'objet d'exception métier de service au composant.

Exceptions d'exécution

Une exception d'exécution peut se produire lorsqu'il existe un problème de communication avec le service. Le traitement d'une exception d'exécution est identique au traitement d'une exception métier. Si un sélecteur d'erreurs a été configuré, le traitement suivant se produit :

1. La liaison d'importation appelle le gestionnaire de données d'exception d'exécution approprié avec les données d'exception.
2. Le gestionnaire de données d'exception d'exécution transforme les données d'exception en un objet d'exception d'exécution de service et le renvoie à la liaison d'importation.
3. L'importation renvoie l'objet d'exception d'exécution de service au composant.

Interopérabilité entre les modules SCA et les services Open SCA

IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture (SCA) offre un modèle de programmation simple, mais puissant, permettant de créer des applications basées sur les spécifications Open SCA. Les modules SCA de WebSphere Process Server utilisent des liaisons d'importation et d'exportation pour interagir avec les services Open SCA développés dans un environnement Rational Application Developer et hébergés par WebSphere Application Server Feature Pack for Service Component Architecture.

Une application SCA appelle une application Open SCA par l'intermédiaire d'une liaison d'importation. Une application SCA reçoit un appel d'une application Open SCA par l'intermédiaire d'une liaison d'exportation. Une liste des liaisons prises en charge est présentée dans «Appel de services sur des liaisons interopérables», à la page 34.

Appel de services Open SCA à partir de modules SCA

Les applications SCA développées avec WebSphere Integration Developer peuvent appeler des applications Open SCA développées dans un environnement Rational Application Developer. Cette section illustre un exemple d'appel d'un service Open SCA à partir d'un module SCA à l'aide d'une liaison d'importation SCA.

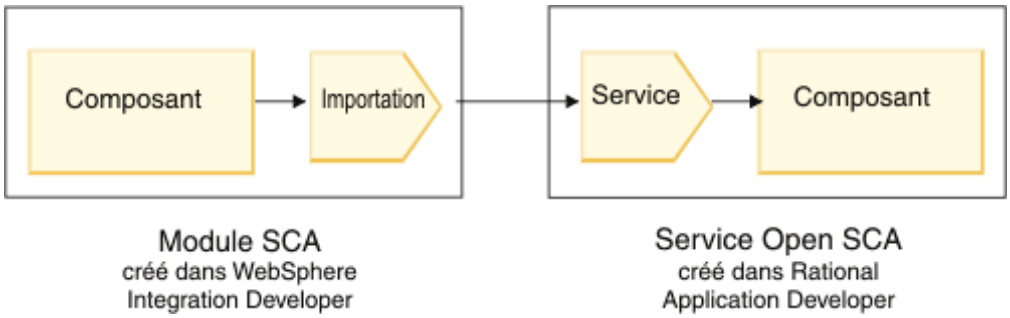


Figure 12. Composant de module SCA appelant un service Open SCA

Aucune configuration spéciale n'est requise pour appeler un service Open SCA.

Pour vous connecter à un service Open SCA par l'intermédiaire d'une liaison d'importation SCA, spécifiez le nom de composant et le nom de service du service Open SCA dans la liaison d'importation.

1. Pour obtenir le nom du composant cible et du service à partir du composite Open SCA, procédez comme suit :
   a. Vérifiez que la page **Propriétés** est ouverte en cliquant sur **Fenêtre** → **Afficher la vue** → **Propriétés**.
   b. Ouvrez l'éditeur de composite en cliquant deux fois sur le diagramme de composite qui contient le composant et le service. Par exemple, pour un composant intitulé **customer**, le diagramme du composite est **customer.composite_diagram**.
   c. Cliquez sur le composant cible.
   d. Dans la zone **Nom** de la page **Propriétés**, relevez le nom du composant cible.
   e. Cliquez sur l'icône de service associé au composant.
   f. Dans la zone **Nom** de la page **Propriétés**, relevez le nom du service.
2. Pour configurer l'importation WebSphere Process Server afin de la connecter au service Open SCA, procédez comme suit :
   a. Dans WebSphere Integration Developer, accédez à l'onglet **Propriétés** de l'importation SCA à connecter au service Open SCA.
   b. Dans la zone **Nom du module**, entrez le nom de composant de l'étape 1d.
   c. Dans la zone **Nom d'exportation**, entrez le nom de service de l'étape 1f.
   d. Sauvegardez votre travail en appuyant sur les touches Ctrl+S.

Appel de modules SCA à partir de services Open SCA

Les applications Open SCA développées dans un environnement Rational Integration Developer peuvent appeler des applications SCA développées avec WebSphere Integration Developer. Cette section illustre un exemple d'appel d'un module SCA (par l'intermédiaire d'une liaison d'exportation SCA) à partir d'un service Open SCA.

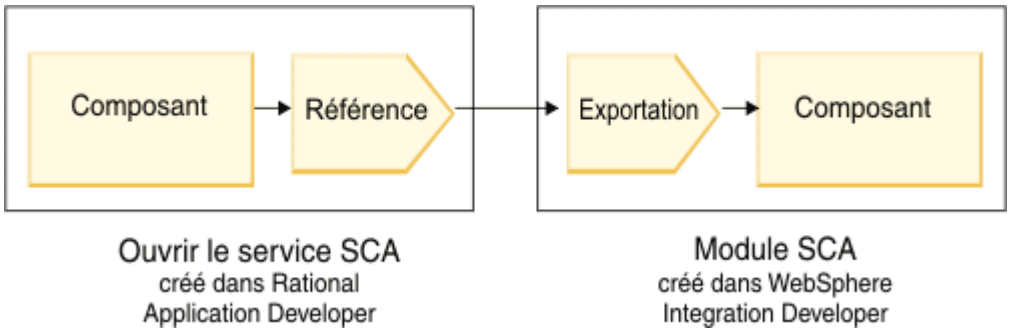


Figure 13. Service Open SCA appelant un composant dans un module SCA

Pour vous connecter à un composant SCA par l'intermédiaire d'une liaison de référence Open SCA, vous spécifiez le nom du module et le nom de l'exportation.

1. Pour obtenir le nom du composant cible et de l'exportation, procédez comme suit :
   a. Dans WebSphere Integration Developer, ouvrez le module dans l'éditeur d'assemblage en cliquant deux fois sur le module.
   b. Cliquez sur l'exportation.
   c. Dans la zone **Nom** de la page **Propriétés**, relevez le nom de l'exportation.
2. Configurez la référence Open SCA à connecter au module WebSphere Process Server et à l'exportation :
   a. Dans Rational Application Developer, ouvrez l'éditeur de composite en cliquant deux fois sur le diagramme de composite qui contient le composant et le service.
   b. Cliquez sur l'icône de référence de la référence de composant pour en afficher les propriétés dans la page **Propriétés**.
   c. Cliquez sur l'onglet **Liaison** dans la partie gauche de la page.
   d. Cliquez sur **Liaisons**, puis sur **Ajouter**.
   e. Sélectionnez la liaison **SCA**.
   f. Dans la zone **Uri**, entrez le nom du module WebSphere Process Server, suivi d'une barre oblique («/»), suivie du nom de l'exportation (que vous avez déterminé à l'étape 1c).
   g. Cliquez sur **OK**.
   h. Sauvegardez votre travail en appuyant sur les touches Ctrl+S.

Appel de services sur des liaisons interopérables

Les liaisons ci-après sont prises en charge pour l'interopérabilité avec un service Open SCA.

- Liaison SCA

  Lorsqu'un module SCA WebSphere Process Server appelle un service Open SCA par l'intermédiaire d'une liaison d'importation SCA, les styles d'appel suivants sont pris en charge :

  – Asynchrone (unidirectionnel)
  – Synchrone (demande/réponse)

  L'interface de l'importation SCA et celle du service Open SCA doivent utiliser une interface WSDL compatible avec l'interopérabilité des services Web (WS-I).

  Notez que la liaison SCA prend en charge la propagation des transactions et des contextes de sécurité.

- Liaison de service Web (JAX-WS) avec protocole SOAP1.1/HTTP ou SOAP1.2/HTTP

  L'interface de l'importation SCA et celle du service Open SCA doivent utiliser une interface WSDL compatible avec l'interopérabilité des services Web (WS-I).

  En outre, les qualités de service suivantes sont prises en charge :

  – Transaction atomique des services Web
  – Sécurité des services Web
- Liaison EJB

  Une interface Java est utilisée pour définir l'interaction entre un module SCA et un service Open SCA lorsque la liaison EJB est utilisée.

  Notez que la liaison EJB prend en charge la propagation des transactions et des contextes de sécurité.
- Liaisons JMS

  L'interface de l'importation SCA et celle du service Open SCA doivent utiliser une interface WSDL compatible avec l'interopérabilité des services Web (WS-I).

  Les fournisseurs JMS pris en charge sont les suivants :

  – WebSphere Platform Messaging (Liaison JMS)
  – WebSphere MQ (Liaison JMS MQ)

Remarque : Les graphiques métier ne sont pas interopérables entre les liaisons SCA et ne sont donc pas prises en charge dans les interfaces utilisées pour interagir avec WebSphere Application Server Feature Pack for Service Component Architecture.

Types de liaison

Vous pouvez utiliser des*liaisons* spécifiques à un protocole avec des importations et des exportations pour spécifier les moyens de transport des données depuis ou vers un module.

Concepts associés

«Sélection des liaisons appropriées», à la page 37
Diverses liaisons sont disponibles pour satisfaire les besoins de votre application.

«Liaisons SCA», à la page 38
Une liaison SCA (Service Component Architecture) permet à un service de communiquer avec d'autres services dans d'autres modules. Une importation avec une liaison SCA permet d'accéder à un service dans un autre module SCA. Une exportation avec une liaison SCA permet d'offrir un service à d'autres modules.

«Liaisons de service Web», à la page 38
Une liaison de service Web représente le moyen par lequel les messages sont transmis d'un composant SCA (Service Component Architecture) vers un service Web (et inversement).

«Liaisons HTTP», à la page 59
La liaison HTTP permet de relier une architecture SOA à HTTP. Par conséquent, les applications HTTP existantes ou récemment développées peuvent être intégrées aux environnements d'architecture SOA (Service Oriented Architecture).

«Liaisons EJB», à la page 69
Les liaisons d'importation EJB (Enterprise JavaBeans permettent aux composants SCA d'appeler les services fournis par la logique métier de Java EE exécutée sur un serveur Java EE. Les liaisons d'exportation EJB permettent aux composants d'être affichés comme des EJB (Enterprise JavaBeans) pour que la logique métier de Java EE puisse appeler les composants SCA qui ne seraient autrement pas disponibles.

«Liaisons EIS», à la page 78
Les liaisons EIS (Enterprise Information System) assurent la connectivité entre les composants SCA et un système d'information d'entreprise externe. Cette communication est accomplie à l'aide des importations et exportations EIS qui prennent en charge les adaptateurs de ressources JCA 1.5 et les adaptateurs Websphere.

«Liaisons JMS», à la page 85
Un fournisseur JMS (Java Message Service) active la messagerie en fonction du modèle de programmation et de l'API JMS (Java Messaging Service). Il fournit des fabriques de connexions Java EE pour créer des connexions pour des destinations JMS et pour envoyer et recevoir des messages.

«Liaisons JMS génériques», à la page 96
La liaison JMS générique assure la connectivité avec les fournisseurs tiers compatibles avec JMS 1.1. Le fonctionnement des liaisons JMS génériques est identique à celui des liaisons JMS.

«Liaisons JMS WebSphere MQ», à la page 104
La liaison JMS WebSphere MQ assure l'intégration avec les applications externes qui utilisent un fournisseur basé sur JMS WebSphere MQ.

«Liaisons WebSphere MQ», à la page 113
La liaison WebSphere MQ assure une connectivité SCA (Service Component Architecture) avec les applications WebSphere MQ.

Information associée

«Limitations des liaisons», à la page 124
L'utilisation des liaisons connaît certaines limitations qui sont répertoriées ici.

Sélection des liaisons appropriées

Diverses liaisons sont disponibles pour satisfaire les besoins de votre application.

Les liaisons disponibles dans WebSphere Integration Developer offrent une vaste plage d'options possibles. Cette liste permet d'identifier le type de liaison le plus approprié pour les besoins de votre application.

Choisissez une liaison *SCA* si les facteurs suivants sont applicables :

- Tous les services se trouvent dans des modules WebSphere Integration Developer ; il n'existe pas de services externes.
- Vous souhaitez séparer les fonctions dans des modules SCA différents qui interagissent directement entre eux.
- Les modules sont étroitement couplés.

Choisissez une liaison de *service Web* si les facteurs suivants sont applicables :

- Vous devez accéder à un service externe via Internet ou fournir un service via Internet.
- Les services sont faiblement couplés.
- Vous préférez les communications synchrones ; une demande d'un service peut attendre une réponse d'un autre service.
- Le protocole du service externe auquel vous accédez ou du service à fournir est SOAP/HTTP ou SOAP/JMS.

Choisissez une liaison *HTTP* si les facteurs suivants sont applicables :

- Vous devez accéder à un service externe via Internet ou fournir un service via Internet et vous utilisez d'autres services Web basés sur le modèle HTTP (vous utilisez des opérations d'interface HTTP connues comme GET, PUT, DELETE, etc.)
- Les services sont faiblement couplés.
- Vous préférez les communications synchrones ; une demande d'un service peut attendre une réponse d'un autre service.

Choisissez une liaison *EJB* si les facteurs suivants sont applicables :

- La liaison est destinée à un service importé qui est lui-même un EJB ou auquel les clients EJB doivent accéder.
- Le service importé est faiblement couplé.
- Les interactions EJB avec état ne sont pas requises.
- Vous préférez les communications synchrones ; une demande d'un service peut attendre une réponse d'un autre service.

Choisissez une liaison *EIS* si les facteurs suivants sont applicables :

- Vous devez accéder à un service sur un système EIS à l'aide d'un adaptateur de ressources.
- Vous préférez une transmission de données synchrone à une transmission asynchrone.

Choisissez une liaison *JMS* si les facteurs suivants sont applicables :

Remarque : Il existe plusieurs types de liaison JMS. Si vous prévoyez d'échanger des messages SOAP à l'aide de JMS, choisissez la liaison de service Web avec le protocole SOAP/JMS. Voir «Liaisons de service Web», à la page 38.

- Vous devez accéder à un système de messagerie.
- Les services sont faiblement couplés.
- Vous préférez une transmission de données asynchrone à une transmission synchrone.

Choisissez une liaison *JMS générique* si les facteurs suivants sont applicables :

- Vous devez accéder à un système de messagerie d'un fournisseur non IBM.
- Les services sont faiblement couplés.
- La fiabilité est plus importante que les performances ; vous préférez une transmission de données asynchrone à une transmission synchrone.

Choisissez une liaison *MQ* si les facteurs suivants sont applicables :

- Vous devez accéder à un système de messagerie WebSphere MQ et utiliser les fonctions natives de MQ.
- Les services sont faiblement couplés.
- La fiabilité est plus importante que les performances ; vous préférez une transmission de données asynchrone à une transmission synchrone.

Choisissez une liaison *JMS MQ* si les facteurs suivants sont applicables :

- Vous devez accéder à un système de messagerie WebSphere MQ, mais pouvez le faire dans un contexte JMS ; le sous-ensemble JMS de fonctions est suffisant pour votre application.
- Les services sont faiblement couplés.
- La fiabilité est plus importante que les performances ; vous préférez une transmission de données asynchrone à une transmission synchrone.

Liaisons SCA

Une liaison SCA (Service Component Architecture) permet à un service de communiquer avec d'autres services dans d'autres modules. Une importation avec une liaison SCA permet d'accéder à un service dans un autre module SCA. Une exportation avec une liaison SCA permet d'offrir un service à d'autres modules.

Vous utilisez WebSphere Integration Developer pour générer et configurer des liaisons SCA pour les importations et les exportations dans vos modules SCA.

Si des modules sont exécutés sur le même serveur ou déployés dans le même cluster, une liaison SCA représente la liaison la plus simple et la plus rapide à utiliser.

Une fois que le module contenant la liaison SCA est déployé sur le serveur, vous pouvez utiliser la console d'administration pour afficher des informations sur la liaison ou, dans le cas d'une liaison d'importation, pour changer les propriétés sélectionnées de cette liaison.

Liaisons de service Web

Une liaison de service Web représente le moyen par lequel les messages sont transmis d'un composant SCA (Service Component Architecture) vers un service Web (et inversement).

Concepts associés

«Présentation des liaisons de service Web»
Une liaison d'importation de service Web vous permet d'appeler un service Web externe depuis vos composants SCA. Une liaison d'exportation de service Web vous permet d'exposer vos composants SCA aux clients en tant que services Web.

«Propagation d'en-têtes SOAP», à la page 40
Lors de la gestion des messages SOAP, vous devez accéder aux informations de certains en-têtes SOAP dans les messages reçus. Vérifiez que des messages avec des en-têtes SOAP sont envoyés avec des valeurs spécifiques ou autorisez ces en-têtes SOAP à passer par un module.

«Pièces jointes dans les messages SOAP», à la page 44
Vous pouvez envoyer et recevoir des messages SOAP qui incluent des données binaires (fichiers PDF ou images JPEG) comme pièces jointes. Les pièces jointes peuvent être *référencées* (à savoir, représentées explicitement comme parties de message dans l'interface de service) ou *non référencées* (auquel cas, un nombre et des types arbitraires de pièces jointes peuvent être incluses).

«Utilisation d'une liaison de style Document WSDL avec des message composite», à la page 58
L'organisation WS-I (Web Services Interoperability Organization) a défini un ensemble de règles sur la manière dont les services Web doivent être décrits par l'intermédiaire d'un WSDL et dont les messages SOAP correspondants doivent être constitués, pour assurer l'interopérabilité.

Présentation des liaisons de service Web

Une liaison d'importation de service Web vous permet d'appeler un service Web externe depuis vos composants SCA. Une liaison d'exportation de service Web vous permet d'exposer vos composants SCA aux clients en tant que services Web.

Avec une liaison de service Web, vous pouvez accéder à des services externes via des messages SOAP interopérables et des qualités de service (QoS).

Vous utilisez WebSphere Integration Developer pour générer et configurer des liaisons de service Web pour les importations et les exportations dans vos modules SCA. Les types de liaison de service Web suivants sont disponibles :

- SOAP1.2/HTTP et SOAP1.1/HTTP

  Ces liaisons sont basées sur JAX-WS (Java API for XML Web Services), une API de programmation Java permettant de créer des services Web.

  – Utilisez SOAP1.2/HTTP si votre service Web respecte la spécification SOAP 1.2.
  – Utilisez SOAP1.1/HTTP si votre service Web respecte la spécification SOAP 1.1.

  Lorsque vous sélectionnez l'une de ces liaisons, vous pouvez envoyer des pièces jointes avec vos messages SOAP.

  Les liaisons de service Web fonctionnent avec les messages SOAP standard. Toutefois, à l'aide de l'une des liaisons JAX-WS de service Web, vous pouvez personnaliser la manière dont les messages SOAP sont analysés ou écrits. Par exemple, vous pouvez gérer les éléments non standard dans des messages SOAP ou appliquer un traitement supplémentaire au message SOAP. Lorsque vous configurez la liaison, vous spécifiez un gestionnaire de données personnalisé qui effectue ce traitement sur le message SOAP.
- SOAP1.1/HTTP

Utilisez cette liaison si vous souhaitez créer des services Web qui utilisent un message codé par SOAP en fonction de la spécification JAX-RPC (Java API for XML-based RPC).

- SOAP1.1/JMS

  Utilisez cette liaison pour envoyer ou recevoir des messages SOAP à l'aide d'une destination JMS (Java Message Service).

Quel que soit le transport (HTTP ou JMS) utilisé pour transmettre le message SOAP, les liaisons de service Web traitent toujours les interactions de demande/réponse de manière synchrone. L'unité d'exécution à l'origine de l'appel sur le fournisseur de services est bloquée jusqu'à la réception d'une réponse du fournisseur. Pour plus d'informations sur ce style d'appel, voir «Appel synchrone».

Important : Les combinaisons de liaisons de service Web suivantes ne peuvent pas être utilisées sur les les exportations du même module. Si vous devez exposer des composants à l'aide de plusieurs de ces liaisons d'exportation, vous devez placer chacun dans un module distinct, puis connecter ces modules à vos composants à l'aide de la liaison SCA :

- SOAP 1.1/JMS et SOAP 1.1/HTTP à l'aide de JAX-RPC
- SOAP 1.1/HTTP à l'aide de JAX-RPC et SOAP 1.1/HTTP à l'aide de JAX-WS
- SOAP 1.1/HTTP à l'aide de JAX-RPC et SOAP 1.2/HTTP à l'aide de JAX-WS

Une fois que le module SCA contenant la liaison de service Web est déployé sur le serveur, vous pouvez utiliser la console d'administration pour afficher des informations sur la liaison ou pour changer les propriétés sélectionnées de cette liaison.

Remarque : Les services Web permettent aux applications d'interopérer en utilisant les descriptions standard des services et les formats standard des messages qu'ils échangent. Par exemple, les liaisons d'importation et d'exportation de services Web peuvent interopérer avec des services implémentés au moyen de Web Services Enhancements (WSE) Version 3.5 et Windows Communication Foundation (WCF) Version 3.5 pour Microsoft.NET. Lors de l'interopération avec de tels services, vous devez veiller à ce que les conditions suivantes soient remplies :

- Le fichier WSDL (Web Services Description Language) utilisé pour accéder à une exportation de service Web inclut une valeur d'action SOAP non vide pour chaque opération exposée dans l'interface.
- Le client de service Web (WSC) définit l'en-tête SOAPAction ou wsa:Action lors de l'envoi de messages à une exportation de service Web.

Propagation d'en-têtes SOAP

Lors de la gestion des messages SOAP, vous devez accéder aux informations de certains en-têtes SOAP dans les messages reçus. Vérifiez que des messages avec des en-têtes SOAP sont envoyés avec des valeurs spécifiques ou autorisez ces en-têtes SOAP à passer par un module.

Lorsque vous configurez une liaison de services Web dans WebSphere Integration Developer, vous pouvez indiquer que les en-têtes SOAP doivent être propagés.

- Lorsque des demandes sont reçues lors d'une exportation ou des réponses sont reçues lors d'une importation, les informations sur les en-têtes SOAP sont accessibles, ce qui permet à la logique dans un module de se baser sur les valeurs d'en-têtes et à ces en-têtes d'être modifiés.

- Lorsque des demandes sont envoyées d'une exportation ou des réponses sont envoyées d'une importation, des en-têtes SOAP peuvent être inclus dans ces messages.

La forme et la présence des en-têtes SOAP propagés peuvent être affectées par les ensembles de règles configurés dans l'importation ou l'exportation, comme décrit dans le tableau 9, à la page 42.

Pour configurer la propagation des en-têtes SOAP pour une importation ou une exportation, vous cliquez (dans la vue Propriétés de WebSphere Integration Developer) dans l'onglet **Propager l'en-tête de protocole** et sélectionnez les options dont vous avez besoin.

En-tête WS-Addressing

L'en-tête WS-Addressing peut être propagé par la liaison de service Web (JAX-WS).

Quand vous propagez l'en-tête WS-Addressing, prenez en compte les informations suivantes :

- Si vous activez la propagation pour l'en-tête WS-Addressing, celui-ci est propagé dans le module dans les cas suivants :
  - si des demandes sont reçues dans une exportation,
  - si des réponses sont reçues dans une importation.
- L'en-tête WS-Addressing n'est pas propagé dans des messages sortants de WebSphere Process Server (l'en-tête n'est pas propagé quand des demandes sont envoyées d'une importation ou quand des réponses sont envoyées d'une exportation).

En-tête WS-Security

L'en-tête WS-Security peut être propagé par la liaison de service Web (JAX-WS) et la liaison de service Web (JAX-RPC).

La spécification WS-Security des services Web décrit les améliorations apportées à la messagerie SOAP pour garantir la qualité de protection via l'intégrité des messages, la confidentialité des messages et leur authentification unique. Ces mécanismes peuvent permettre d'accepter toute une gamme de modèles de sécurité et de technologies de chiffrement.

Quand vous propagez l'en-tête WS-Security, prenez en compte les informations suivantes :

- Si vous activez la propagation pour l'en-tête WS-Security, celui-ci est propagé à travers le module dans les cas suivants :
  - si des demandes sont reçues dans une exportation,
  - si des demandes sont envoyées d'une importation,
  - si des réponses sont reçues dans une importation.
- Par défaut, l'en-tête n'est *pas* propagé quand des réponses sont envoyées d'une exportation Toutefois, si vous définissez la propriété JVM `WSSECURITY.ECHO.ENABLED` à `true`, l'en-tête est propagé quand des réponses sont envoyées de l'exportation. Dans ce cas, si l'en-tête WS-Security dans le chemin de demande n'est pas modifié, les en-têtes WS-Security peuvent être automatiquement répercutés de demandes en réponses.
- La forme exacte du message SOAP envoyé d'une importation pour une demande ou d'une exportation pour une réponse peut ne pas exactement correspondre au

message SOAP reçu à l'origine. C'est pourquoi toute signature numérique est censée devenir invalide. Si une signature numérique est obligatoire dans les messages envoyés, elle doit être établie à l'aide d'un ensemble de règles de sécurité approprié, et les en-têtes relatifs à la signature numérique dans les messages reçus doivent être supprimés dans le module.

Pour propager l'en-tête WS-Security, vous devez inclure le schéma WS-Security avec le module d'application. Voir «Inclusion du schéma WS-Security dans un module d'application», à la page 43 pour connaître la procédure pour inclure le schéma.

Mode de propagation des en-têtes

Le mode de propagation des en-têtes dépend de la définition des règles de sécurité sur la liaison d'importation ou d'exportation, comme décrit dans le tableau 9 :

Tableau 9. Mode de transmission des en-têtes de sécurité

	Liaison d'exportation sans règle de sécurité	**Liaison d'exportation avec règle de sécurité**
Liaison d'importation sans règle de sécurité	Les en-têtes de sécurité sont transmis en l'état via le module. Ils ne sont pas déchiffrés. Les en-têtes sont envoyés en sortie sous la même forme qu'ils ont été reçus. La signature numérique peut devenir invalide.	Les en-têtes de sécurité sont déchiffrés et transmis via le module avec la vérification et l'authentification de la signature. Les en-têtes déchiffrés sont envoyés en sortie. La signature numérique peut devenir invalide.
Liaison d'importation avec règle de sécurité	Les en-têtes de sécurité sont transmis en l'état via le module. Ils ne sont pas déchiffrés. Les en-têtes ne doivent pas être propagés vers l'importation. Sinon, une erreur se produit en raison de la duplication.	Les en-têtes de sécurité sont déchiffrés et transmis via le module avec la vérification et l'authentification de la signature. Les en-têtes ne doivent pas être propagés vers l'importation. Sinon, une erreur se produit en raison de la duplication.

Configurez les ensembles de règles appropriés sur les liaisons d'exportation et d'importation, car l'opération épargne le demandeur de service des modifications apportées à la configuration ou aux exigences QoS du fournisseur de services. Si des en-têtes SOAP standard sont visibles dans un module, ils peuvent être utilisés pour influencer le traitement (par exemple, la consignation et le traçage) dans le module. La propagation des en-têtes SOAP à travers le module, d'un message reçu à un message envoyé, signifie que les avantages de l'isolement du module sont moindres.

Les en-têtes standard, comme les en-têtes WS-Security, ne doivent pas être propagés sur une demande vers une importation ou sur une réponse vers une exportation lorsqu'un ensemble de règles, qui entraînerait normalement la génération de ces en-têtes, est associé à une importation ou une exportation. Sinon, une erreur se produit en raison d'une duplication de ces en-têtes. A la place, les en-têtes doivent être explicitement supprimés, ou bien la liaison d'importation ou

d'exportation doit être configurée pour empêcher la propagation des en-têtes de protocole.

Accès aux en-têtes SOAP

Quand un message contenant des en-têtes est reçu d'une importation ou d'une exportation de service Web, les en-têtes sont placés dans la section d'en-têtes de l'objet de message de service (SMO). Vous pouvez accéder aux informations d'en-tête, comme décrit dans «Accès aux informations d'en-tête SOAP dans le SMO».

Inclusion du schéma WS-Security dans un module d'application

La procédure suivante montre les étapes pour inclure le schéma dans le module d'application :

- Si l'ordinateur sur lequel WebSphere Integration Developer s'exécute a accès à Internet, procédez comme suit :
  1. Dans la perspective Intégration métier, sélectionnez **Dépendances** pour votre projet.
  2. Développez **Ressources prédéfinies** et sélectionnez **Fichiers de schéma WS-Security 1.0** ou **Fichiers de schéma WS-Security 1.1** pour importer le schéma dans votre module.
  3. Effacez et régénérez le projet.
- Si l'ordinateur sur lequel WebSphere Integration Developer s'exécute n'a pas accès à Internet, vous pouvez télécharger le schéma sur un autre ordinateur ayant accès à Internet. Vous pouvez ensuite le copier sur l'ordinateur où WebSphere Integration Developer s'exécute.
  1. Depuis l'ordinateur qui a accès à Internet, téléchargez le schéma distant :
     a. Cliquez sur **Fichier** → **Importer** → **Intégration métier** → **WSDL et XSD**.
     b. Sélectionnez **WSDL distant** ou **Fichier XSD**.
     c. Importez les schémas suivants :

        http://www.w3.org/2003/05/soap-envelope/

        http://www.w3.org/TR/2002/REC-xmlenc-core-20021210 /xenc-schema.xsd

        http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd
  2. Copiez les schémas sur l'ordinateur sans accès à Internet.
  3. Depuis l'ordinateur sans accès à Internet, importez le schéma :
     a. Cliquez sur **Fichier** → **Importer** → **Intégration métier** → **WSDL et XSD**.
     b. Sélectionnez **WSDL local** ou **Fichier XSD**.
  4. Modifiez les emplacements des schémas pour oasis-wss-wssecurity_secext-1.1.xsd :
     a. Ouvrez le schéma dans*emplacement_espace_de_travail/nom_module/*`StandardImportFilesGen/oasis-wss-wssecurity-secext-1.1.xsd`.
     b. Remplacez :

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='http://www.w3.org/2003/05/soap-envelope/'/>
        ```

        par :

        ```
        <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
        schemaLocation='../w3/_2003/_05/soap_envelope.xsd'/>
        ```

     c. Remplacez :

```
<xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
schemaLocation='http://www.w3.org/TR/2002
/REC-xmlenc-core-20021210/xenc-schema.xsd'/>
```

par :

```
<xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
schemaLocation='../w3/tr/_2002/rec_xmlenc_core_20021210/xenc-schema.xsd'/>
```

5. Modifiez l'emplacement du schéma pour oasis-200401-wss-wssecurity-secext-1.0.xsd :
   a. Ouvrez le schéma dans *emplacement_espace_de_travail*/*nom_module*/StandardImportFilesGen/oasis-200401-wss-wssecurity-secext-1.0.xsd.
   b. Remplacez :

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.w3.org/TR/xmldsig-core
/xmldsig-core-schema.xsd"/>
```

par :

```
<xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="../w3/tr/_2002/rec_xmldsig_core
_20020212/xmldsig-core-schema.xsd"/>
```

6. Effacez et régénérez le projet.

Pièces jointes dans les messages SOAP

Vous pouvez envoyer et recevoir des messages SOAP qui incluent des données binaires (fichiers PDF ou images JPEG) comme pièces jointes. Les pièces jointes peuvent être *référencées* (à savoir, représentées explicitement comme parties de message dans l'interface de service) ou *non référencées* (auquel cas, un nombre et des types arbitraires de pièces jointes peuvent être incluses).

Une pièce jointe référencée peut être représentée de l'une des manières suivantes :

- Comme élément wsi:swaRef-typed dans le schéma du message

  Les pièces jointes définies à l'aide du type wsi:swaRef sont conformes à Web Services Interoperability Organization (WS-I) *Attachments Profile Version 1.0* (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), qui définit la manière dont les éléments de message sont associés aux composants MIME.
- Comme composant de message de niveau supérieur, à l'aide d'un schéma de type binaire

  Les pièces jointes représentées sous forme de composants de message de niveau supérieur respectent la spécification des *messages SOAP avec pièces jointes* (http://www.w3.org/TR/SOAP-attachments).

Une pièce jointe non référencée est transportée dans un message SOAP sans aucune représentation dans le schéma du message.

Dans tous les cas, la liaison SOAP WSDL doit inclure une liaison MIME pour que des pièces jointes puissent être utilisées et la taille maximale de ces pièces jointes ne doit pas dépasser 20 Mo.

Remarque : Pour envoyer ou recevoir des messages SOAP avec des pièces jointes, vous devez utiliser l'une des liaisons de service Web basée sur JAX-WS (Java API for XML Web Services).

Concepts associés

«Pièces jointes référencées : éléments de type swaRef»
Vous pouvez envoyer et recevoir des messages SOAP qui incluent des pièces jointes représentées dans l'interface des services comme des éléments de type swaRef.

«Pièces jointes référencées : composants de message de niveau supérieur», à la page 50
Vous pouvez envoyer et recevoir des messages SOAP qui incluent des pièces jointes binaires déclarées comme composants dans votre interface de service.

«Pièces jointes non référencées», à la page 55
Vous pouvez envoyer et recevoir des pièces jointes *non référencées* qui ne sont pas déclarées dans l'interface de service.

Pièces jointes référencées : éléments de type swaRef :

Vous pouvez envoyer et recevoir des messages SOAP qui incluent des pièces jointes représentées dans l'interface des services comme des éléments de type swaRef.

Un élément typé swaRef est défini dans Web Services Interoperability Organization (WS-I) *Attachments Profile* Version 1.0 (http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html), qui définit la manière dont les éléments de message sont associés aux composants MIME.

Remarque : Les messages SOAP générés ou consommés ne sont pas toujours compatibles avec WS-I Attachments Profile. En particulier, le «codage de la portion content-id,» tel qu'il est décrit dans la section 3.8 de WS-I Attachments Profile 1.0, n'est pas pris en charge.

Dans le message SOAP, le corps du message SOAP contient un élément de type swaRef qui identifie l'ID contenu de la pièce jointe.

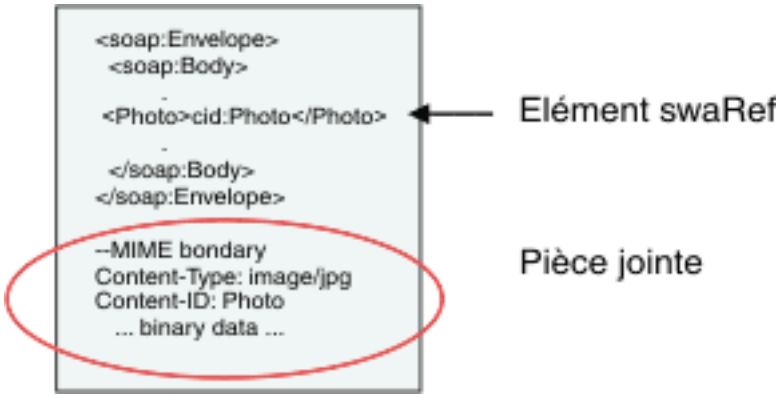


Figure 14. Message SOAP avec élément swaRef

Le WSDL de ce message SOAP contient un élément de type swaRef dans une portion de message qui identifie la pièce jointe.

```
<element name="sendPhoto">
  <complexType>
     <sequence>
       <element name="Photo" type="wsi:swaRef"/>
     </sequence>
  </complexType>
 </element>
```

Le WSDL doit également contenir une liaison MIME qui indique que les messages MIME composite doivent être utilisés.

Remarque : Le WSDL n'inclut *pas* de liaison MIME pour l'élément de message typé swaRef spécifique, car les liaisons MIME ne s'appliquent qu'aux portions de message de niveau supérieur.

La propagation des pièces jointes représentées comme des éléments swaRef-typed n'est possible qu'entre composants de flux de médiation. Si un autre type de composant doit accéder à la pièce jointe ou servir de cible lors de la propagation de celle-ci, utilisez un composant de flux de médiation pour déplacer la pièce jointe vers un emplacement accessible par le composant.

Traitement entrant des pièces jointes

Utilisez WebSphere Integration Developer pour configurer une liaison d'exportation afin de recevoir la pièce jointe. Vous créez un module, ainsi que son interface et ses opérations associées, et notamment un élément de type swaRef. Vous créez ensuite une liaison de service Web (JAX-WS).

Remarque : Pour obtenir des informations détaillées, reportez-vous à la rubrique «Utilisation des pièces jointes» du centre de documentation de WebSphere Integration Developer.

Lorsqu'un client transmet un message SOAP avec une pièce jointe swaRef à un composant SCA (Service Component Architecture), la liaison d'exportation de service Web (JAX-WS) commence par supprimer cette pièce jointe. Elle analyse ensuite la partie SOAP du message et crée un objet métier. Enfin, la liaison définit l'ID contenu de la pièce jointe dans l'objet métier.

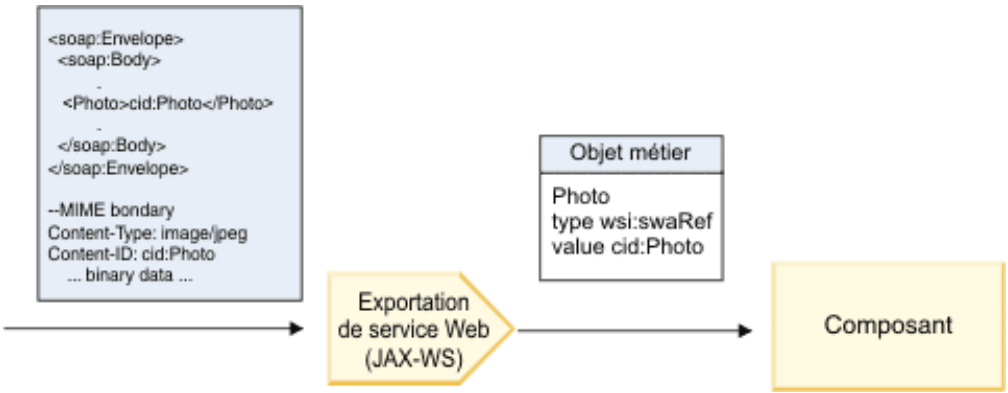


Figure 15. Méthode de traitement d'un message SOAP avec une pièce jointe swaRef par la liaison d'exportation de service Web (JAX-WS)

Accès aux métadonnées de pièce jointe dans un composant de flux de médiation

Comme illustré dans la figure 16, à la page 47, lorsque des composants accèdent à des pièces jointes swaRef, l'identificateur de contenu de la pièce jointe apparaît comme élément de type swaRef.

Chaque pièce jointe d'un message SOAP contient également un élément `attachments` correspondant dans l'objet SMO. Lorsqu'il utilise le type WS-I swaRef, l'élément `attachments` inclut le type de contenu de la pièce jointe et l'ID contenu, ainsi que les données binaires de la pièce jointe.

Pour obtenir la valeur d'une pièce jointe swaRef, il est donc nécessaire d'obtenir la valeur de l'élément typé swaRef, puis de rechercher l'élément `attachments` avec la valeur `contentID` correspondante. Notez que pour la valeur `contentID`, le préfixe `cid:` est généralement supprimé de la valeur swaRef.

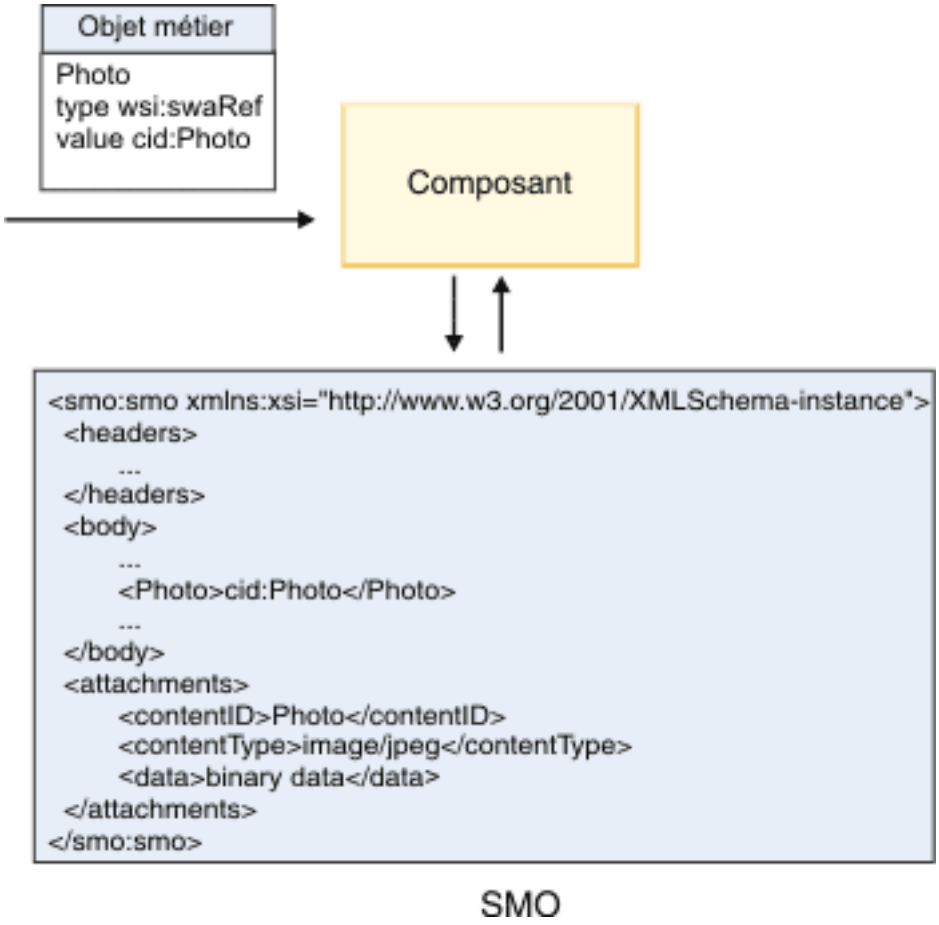


Figure 16. Affichage des pièces jointes swaRef dans l'objet SMO

Traitement sortant

Vous utilisez WebSphere Integration Developer pour configurer une liaison d'importation de service Web (JAX-WS) de sorte qu'elle appelle un service Web externe. La liaison d'importation est configurée avec un document WSDL qui décrit le service Web à appeler et définit la pièce jointe qui sera transmise au service Web.

Lorsqu'un message SCA est reçu par une liaison d'importation de service Web (JAX-WS), les éléments typés swaRef sont envoyés comme pièces jointes si l'importation est connectée à un composant de flux de médiation et l'élément typé swaRef possède un élément `attachments` correspondant.

Pour le traitement sortant, les éléments typés swaRef sont toujours envoyés avec leur ID contenu, mais le module de médiation doit s'assurer qu'il existe un élément `attachments` correspondant avec une valeur `contentID` correspondante.

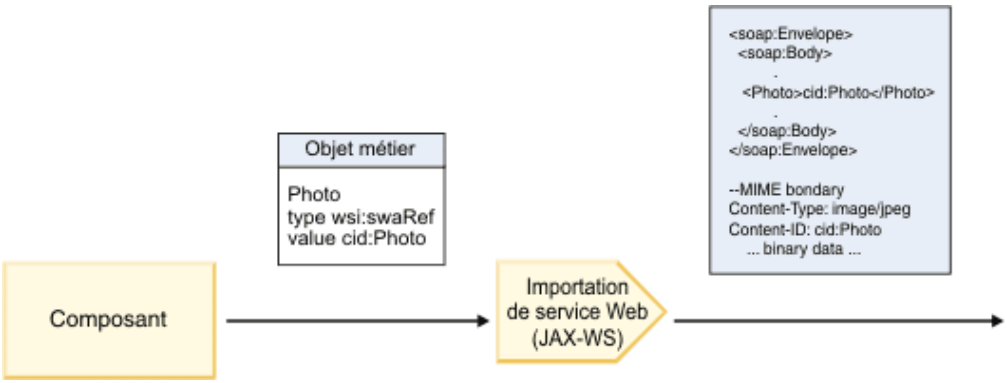


Figure 17. Méthode de génération d'un message SOAP avec une pièce jointe swaRef par la liaison d'importation de service Web (JAX-WS)

Définition des métadonnées de pièce jointe dans un composant de flux de médiation

Si, dans l'objet SMO, il existe une valeur d'élément typé swaRef et un élément `attachments`, la liaison prépare le message SOAP (avec la pièce jointe) et l'envoie à un destinataire.

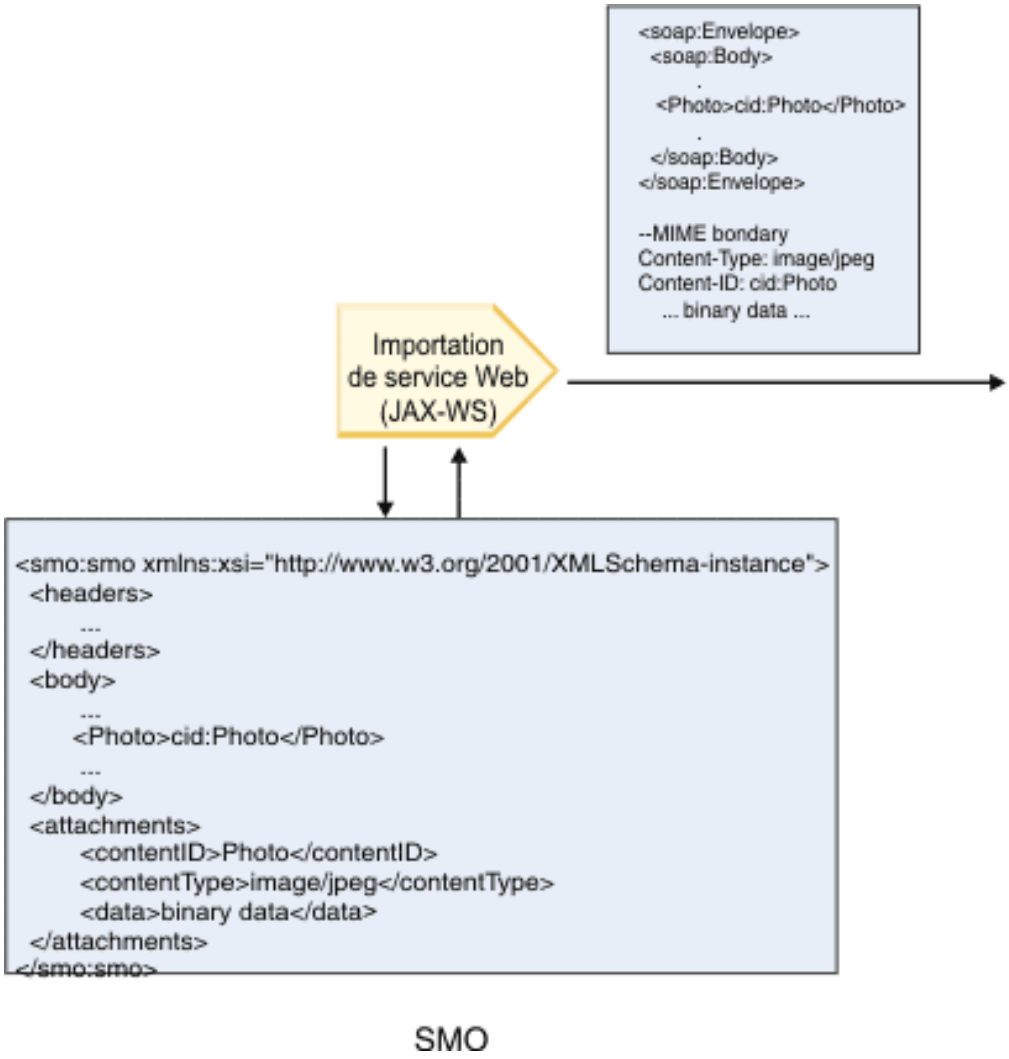


Figure 18. Méthode d'accès à une pièce jointe swaRef de l'objet SMO pour créer le message SOAP

L'élément `attachments` n'est présent dans l'objet SMO que si un composant de flux de médiation est connecté directement à l'importation ou l'exportation ; il n'est pas transmis entre les autres types de composant. Si les valeurs sont requises dans un module qui contient d'autres types de composant, un composant de flux de médiation doit être utilisé pour copier les valeurs dans un emplacement où elles sont accessibles dans le module et un autre composant de flux de médiation doit être utilisé pour définir les valeurs correctes avant un appel sortant par l'intermédiaire d'une importation de service Web.

Important : Comme décrit dans la «représentation XML de l'objet SMO,» la primitive de médiation de transformation XSL convertit les messages à l'aide d'une transformation XSLT 1.0. La transformation agit sur une sérialisation XML de l'objet SMO. La primitive de médiation de transformation XSL permet de spécifier la racine de la sérialisation et l'élément racine du document XML reflète cette racine.

Lorsque vous envoyez des messages SOAP comportant des pièces jointes, l'élément racine choisi détermine le mode de propagation des pièces jointes.

- Si vous utilisez «/body» comme racine de la mappe XML, toutes les pièces jointes sont propagées dans la mappe par défaut.
- Si vous utilisez «/» comme racine de la mappe, vous pouvez contrôler la propagation des pièces jointes.

Pièces jointes référencées : composants de message de niveau supérieur :

Vous pouvez envoyer et recevoir des messages SOAP qui incluent des pièces jointes binaires déclarées comme composants dans votre interface de service.

Dans ce type de message, le corps SOAP constitue la première partie et les pièces jointes figurent dans les parties suivantes. Les références aux pièces jointes sont incluses dans le corps SOAP.

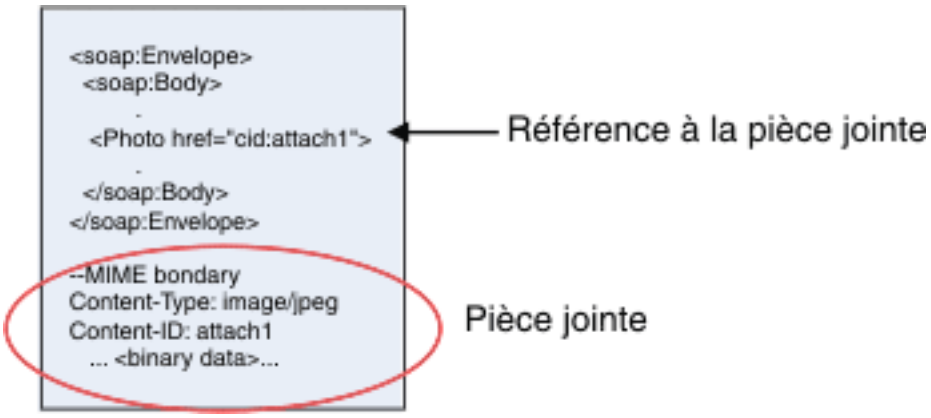


Figure 19. Message SOAP avec une pièce jointe référencée

Quel est l'avantage d'envoyer ou de recevoir une pièce jointe référencée dans un message SOAP ? Les données binaires qui constituent la pièce jointe (dont la taille est souvent importante) sont conservées séparément du corps du message SOAP pour qu'elles n'aient pas besoin d'être analysées comme des données XML. Le traitement est donc plus efficace que si les données binaires étaient conservées dans un élément XML.

Traitement entrant des pièces jointes référencées

Vous pouvez utiliser WebSphere Integration Developer pour configurer la liaison d'exportation. Vous créez un module, ainsi que son interface et ses opérations associées. Vous créez ensuite une liaison de service Web (JAX-WS). La page Pièces jointes référencées affiche toutes les portions binaires de l'opération créée et vous sélectionnez les parties qui seront des pièces jointes.

Remarque : Seules les portions de message de niveau supérieur (les éléments définis dans le type de port WSDL comme portions du message en entrée ou en sortie) dont le type est binaire (base64Binary ou hexBinary) peuvent être envoyées ou reçues en tant que pièces jointes référencées.
Pour obtenir des informations détaillées, reportez-vous à la rubrique «Utilisation des pièces jointes» du centre de documentation de WebSphere Integration Developer.

Lorsqu'un client transmet un message SOAP avec une pièce jointe à un composant SCA (Service Component Architecture), la liaison d'exportation de service Web (JAX-WS) commence par supprimer cette pièce jointe. Elle analyse ensuite la partie

SOAP du message et crée un objet métier. Enfin, la liaison définit le code binaire de la pièce jointe dans l'objet métier.

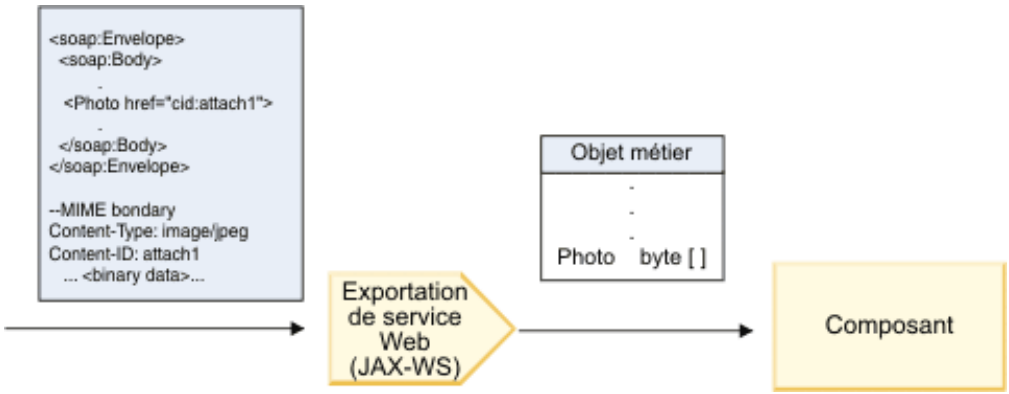


Figure 20. Méthode de traitement d'un message SOAP avec une pièce jointe référencée par la liaison d'exportation de service Web (JAX-WS)

Accès aux métadonnées de pièce jointe dans un composant de flux de médiation

Comme illustré dans la figure 20, lorsque des composants accèdent à des pièces jointes référencées, les données des pièces jointes apparaissent sous forme de tableau octal.

Chaque pièce jointe référencée d'un message SOAP contient également un élément `attachments` correspondant dans l'objet SMO. L'élément `attachments` inclut le type de contenu de la pièce jointe et le chemin d'accès à l'élément de corps de message dans lequel la pièce jointe est conservée.

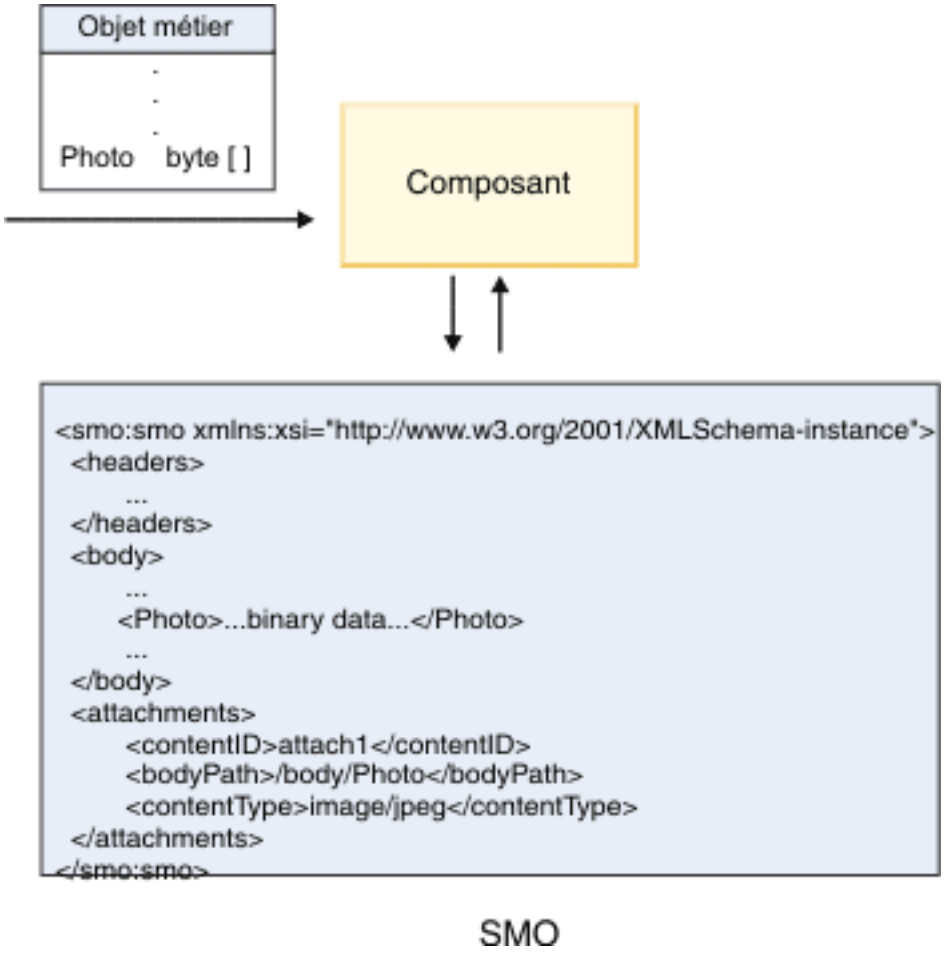


Figure 21. Affichage des pièces jointes référencées dans l'objet SMO

Important : Le chemin d'accès à l'élément de corps du message n'est pas automatiquement mis à jour si le message est converti et la pièce jointe, déplacée. Vous pouvez utiliser un flux de médiation pour mettre à jour l'élément `attachments` avec le nouveau chemin d'accès (par exemple, dans le cadre de la conversion ou à l'aide d'un configurateur d'élément de message distinct).

Traitement sortant des pièces jointes référencées

Vous utilisez WebSphere Integration Developer pour configurer une liaison d'importation de service Web (JAX-WS) de sorte qu'elle appelle un service Web externe. La liaison d'importation est configurée avec un document WSDL qui décrit le service Web à appeler et définit les portions du message qui doivent être transmises comme pièces jointes.

Remarque : La portion qui représente une pièce jointe, telle que définie dans le WSDL, doit être de type simple (base64Binary ou hexBinary). Si une portion de message est définie par un type complexe, elle n'est pas traitée comme pièce jointe.

La liaison d'importation utilise les informations dans le SMO pour déterminer comment les portions de messages binaires de niveau supérieur sont envoyées comme pièces jointes.

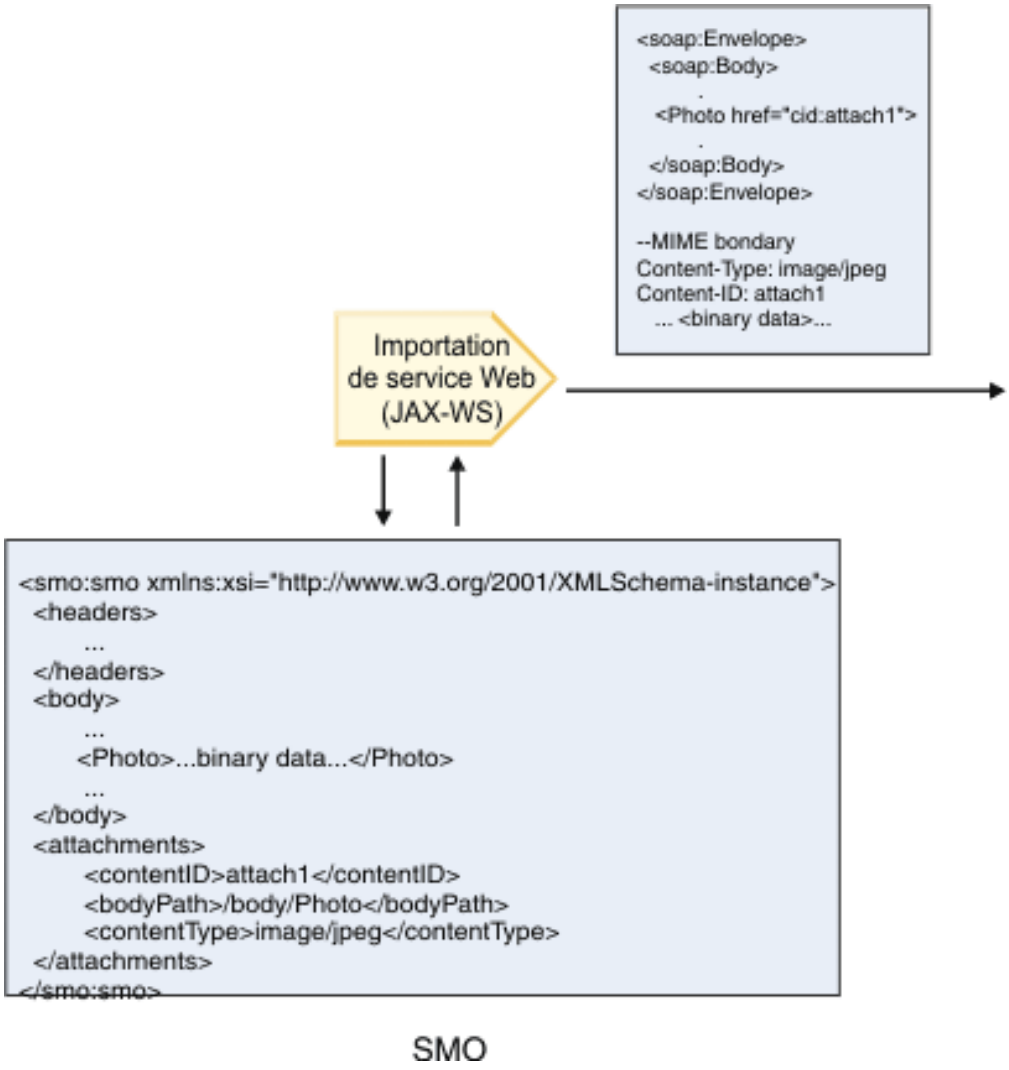


Figure 22. Mode d'accès à la pièce jointe référencée de l'objet SMO pour créer le message SOAP

L'élément `attachments` n'est présent dans l'objet SMO que si un composant de flux de médiation est connecté directement à l'importation ou l'exportation ; il n'est pas transmis entre les autres types de composant. Si les valeurs sont requises dans un module qui contient d'autres types de composant, un composant de flux de médiation doit être utilisé pour copier les valeurs dans un emplacement où elles sont accessibles dans le module et un autre composant de flux de médiation doit être utilisé pour définir les valeurs correctes avant un appel sortant par l'intermédiaire d'une importation de service Web.

La liaison utilise une combinaison des conditions suivantes pour déterminer comment (ou si) le message est envoyé :

- s'il existe une liaison MIME WSDL pour la portion du message binaire de niveau supérieur et, si tel est le cas, comment le type de contenu est défini,
- s'il existe un élément `attachments` dans le SMO dont la valeur `bodyPath` fait référence à une portion binaire de haut niveau,

Mode de création des pièces jointes quand un élément `attachments` se trouve dans le SMO.

Le tableau suivant montre comment une pièce jointe est créée et envoyée si le SMO contient un élément`attachments` avec un élément `bodyPath` correspondant une partie du nom du message :

Tableau 10. Mode de génération de la pièce jointe

Statut de la liaison MIME WSDL pour la portion de message binaire de haut niveau	**Mode de création et d'envoi du message**
Présent avec l'une des situations suivantes : • Aucun type de contenu défini pour la portion du message • Multiple types de contenu définis • Type de contenu avec caractère générique défini	La portion du message est envoyée en tant que pièce jointe. `Content-Id` est définie à la valeur de l'élément attachments le cas échéant ; sinon, une valeur est générée. `Content-Type` est défini à la valeur de l'élément attachments le cas échéant ; sinon, il est défini à `application/octet-stream`.
Présent avec du contenu sans caractère générique pour la portion du message	La portion du message est envoyée en tant que pièce jointe. `Content-Id` est définie à la valeur de l'élément attachments le cas échéant ; sinon, une valeur est générée. `Content-Type` est défini à la valeur de l'élément attachments le cas échéant ; sinon, il est défini au type défini dans l'élément de contenu MIME WSDL.
Non présent	La portion du message est envoyée en tant que pièce jointe. `Content-Id` est définie à la valeur de l'élément attachments le cas échéant ; sinon, une valeur est générée. `Content-Type` est défini à la valeur de l'élément attachments le cas échéant ; sinon, il est défini à `application/octet-stream`. **Remarque :** L'envoi de portions de messages en tant que pièces jointes quand elles ne sont pas définies comme telles dans le langage WSDL peut empêcher la conformité avec WS-I Attachments Profile 1.0 et doit donc être si possible évité.

Mode de création des pièces jointes quand aucun élément `attachments` ne se trouve dans le SMO

Le tableau suivant montre comment une pièce jointe est créée et envoyée si le SMO ne contient pas d'élément `attachments` avec un élément `bodyPath` correspondant à une partie du nom du message :

Tableau 11. Mode de génération de la pièce jointe

Statut de la liaison MIME WSDL pour la portion de message binaire de haut niveau	**Mode de création et d'envoi du message**
Présent avec l'une des situations suivantes : • Aucun type de contenu défini pour la portion du message • Multiple types de contenu définis • Type de contenu avec caractère générique défini	La portion du message est envoyée en tant que pièce jointe. `Content-Id` est généré. `Content-Type` est défini à `application/octet-stream`.
Présent avec du contenu sans caractère générique pour la portion du message	La portion du message est envoyée en tant que pièce jointe. `Content-Id` est généré. `Content-Type` est défini au type défini dans l'élément de contenu MIME WSDL.
Non présent	La portion du Message n'est pas envoyée en tant que pièce jointe.

Important : Comme décrit dans la «représentation XML de l'objet SMO,» la primitive de médiation de transformation XSL convertit les messages à l'aide d'une transformation XSLT 1.0. La transformation agit sur une sérialisation XML de l'objet SMO. La primitive de médiation de transformation XSL permet de spécifier la racine de la sérialisation et l'élément racine du document XML reflète cette racine.

Lorsque vous envoyez des messages SOAP comportant des pièces jointes, l'élément racine choisi détermine le mode de propagation des pièces jointes.

- Si vous utilisez «/body» comme racine de la mappe XML, toutes les pièces jointes sont propagées dans la mappe par défaut.
- Si vous utilisez «/» comme racine de la mappe, vous pouvez contrôler la propagation des pièces jointes.

Pièces jointes non référencées :

Vous pouvez envoyer et recevoir des pièces jointes *non référencées* qui ne sont pas déclarées dans l'interface de service.

Dans ce type de message, le corps SOAP constitue la première partie et les pièces jointes figurent dans les parties suivantes. Aucune référence à la pièce jointe n'est incluse dans le corps SOAP.

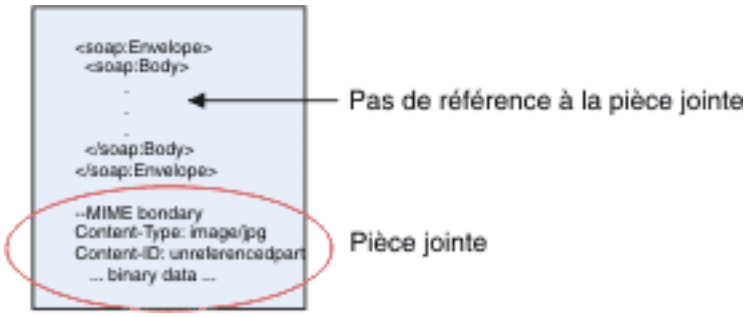


Figure 23. Message SOAP avec une pièce jointe non référencée

Vous pouvez envoyer un message SOAP avec une pièce jointe non référencée par une exportation de service Web vers une importation de service Web. Le message de sortie envoyé au service Web cible contient la pièce jointe.

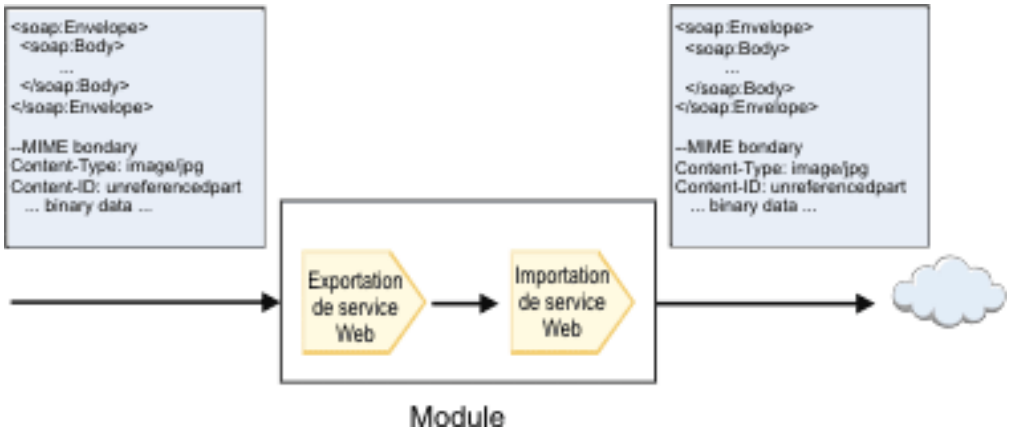


Figure 24. Pièce jointe transitant par un module SCA

Dans la figure 24, le message SOAP comprenant une pièce jointe transite par le module sans modification.

Vous pouvez également modifier le protocole SOAP à l'aide d'un composant de flux de médiation. Par exemple, vous pouvez utiliser le composant de flux de médiation pour extraire des données du message SOAP (dans ce cas, des données binaires dans le corps du message) et créer un message SOAP avec pièces jointes. Les données sont traitées comme partie de l'élément attachments d'un objet de message de service (SMO).

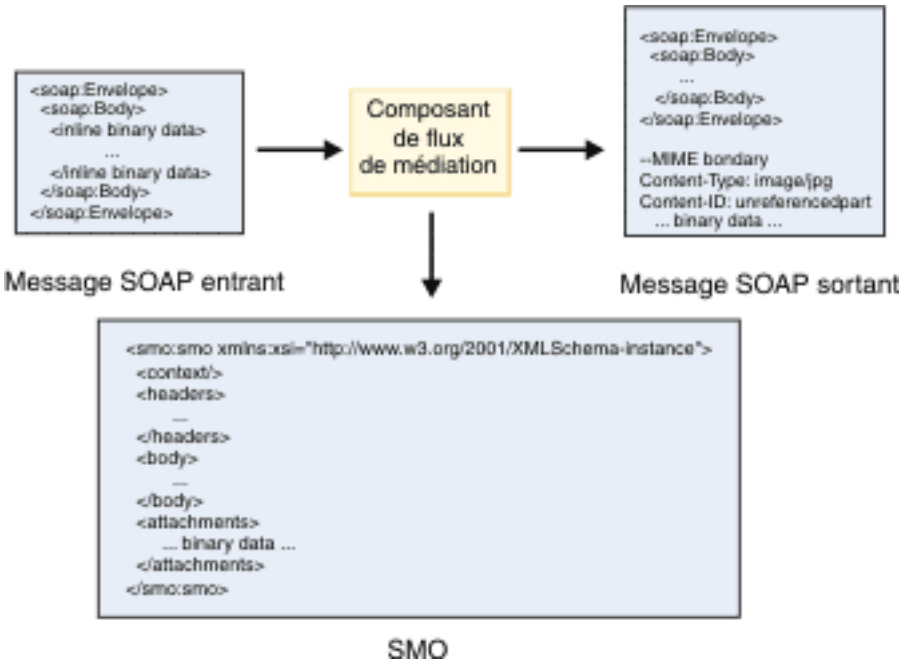


Figure 25. Message traité par un composant de flux de médiation

Inversement, le composant de flux de médiation peut transformer le message entrant en extrayant et en codant la pièce jointe, puis en transmettant le message sans pièce jointe.

Plutôt que d'extraire les données d'un message SOAP entrant pour former un message SOAP avec pièces jointes, vous pouvez obtenir les données de la pièce jointe à partir d'une source externe (par exemple, une base de données).

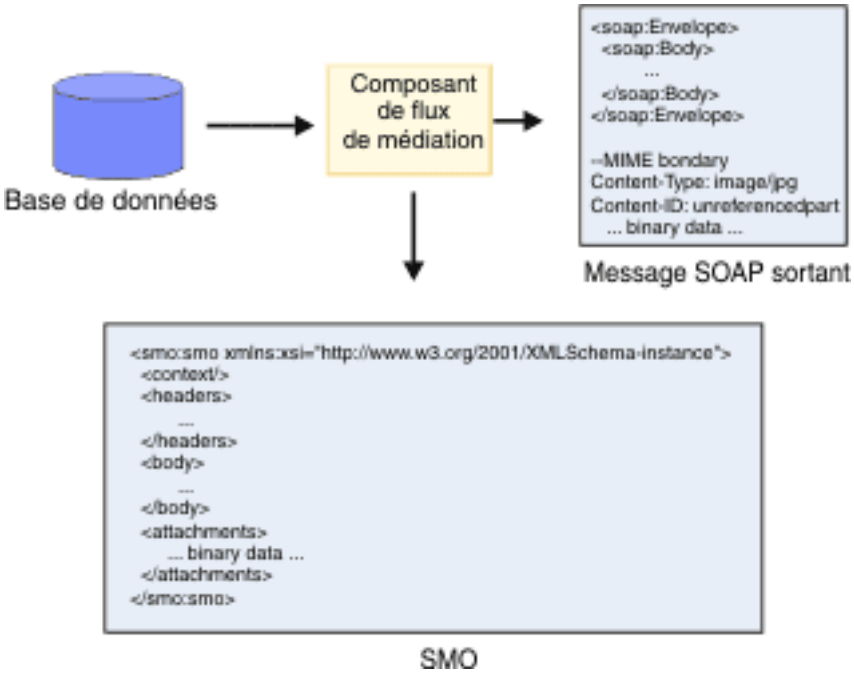


Figure 26. Pièce jointe obtenue d'une base de données et ajoutée au message SOAP

Inversement, le composant de flux de médiation peut extraire la pièce jointe d'un message SOAP entrant et traiter le message (par exemple, stocker la pièce jointe dans une base de données).

La propagation des pièces jointes non référencées n'est possible qu'entre composants de flux de médiation. Si un autre type de composant doit accéder à la pièce jointe ou servir de cible lors de la propagation de celle-ci, utilisez un composant de flux de médiation pour déplacer la pièce jointe vers un emplacement accessible par le composant.

Important : Comme décrit dans la «représentation XML de l'objet SMO,» la primitive de médiation de transformation XSL convertit les messages à l'aide d'une transformation XSLT 1.0. La transformation agit sur une sérialisation XML de l'objet SMO. La primitive de médiation de transformation XSL permet de spécifier la racine de la sérialisation et l'élément racine du document XML reflète cette racine.

Lorsque vous envoyez des messages SOAP comportant des pièces jointes, l'élément racine choisi détermine le mode de propagation des pièces jointes.

- Si vous utilisez «/body» comme racine de la mappe XML, toutes les pièces jointes sont propagées dans la mappe par défaut.
- Si vous utilisez «/» comme racine de la mappe, vous pouvez contrôler la propagation des pièces jointes.

Utilisation d'une liaison de style Document WSDL avec des message composite

L'organisation WS-I (Web Services Interoperability Organization) a défini un ensemble de règles sur la manière dont les services Web doivent être décrits par l'intermédiaire d'un WSDL et dont les messages SOAP correspondants doivent être constitués, pour assurer l'interopérabilité.

Ces règles sont spécifiées dans WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html).

En particulier, pour une liaison SOAP de style document, la compatibilité au profil WS-I requiert que, dans un document WSDL, pour une opération qui utilise le style document, une seule portion de message est associée au corps du message SOAP et que le message SOAP correspondant à cette portion contient un unique élément enfant correspondant à la portion associée de la sorte.

Cela signifie que si une liaison SOAP de style document est utilisée pour une opération dont les messages (entrée, sortie ou incident) sont définis avec plusieurs portions, une seule de ces portions doit être associée au corps du message SOAP pour la compatibilité à WS-I Basic Profile 1.1.

L'approche suivante est utilisée lorsque des descriptions WSDL sont générées pour les exportations avec des liaisons de service Web (JAX-WS et JAX-RPC) dans le cas suivant :

- La première portion de message est associée au corps du message SOAP.
- Pour la liaison JAX-WS, toutes les autres portions de message de type "hexBinary" ou "base64Binary" sont associées comme pièces jointes référencées. Voir «Pièces jointes référencées : composants de message de niveau supérieur», à la page 50.
- Toutes les autres portions de message sont associées comme en-têtes SOAP.

Les liaisons d'importation JAX-RPC et JAX-WS traitent la liaison SOAP d'un document WSDL existant avec des messages de style document composite, même si elle n'associe pas plusieurs portions au corps du message SOAP ; toutefois, vous ne pourrez pas générer de clients de service Web pour de tels documents WSDL dans Rational Application Developer.

Remarque : La liaison JAX-RPC ne prend pas en charge les pièces jointes.

Le modèle recommandé lors de l'utilisation de messages composites avec une opération dont le style de document est une liaison SOAP est donc le suivant :

1. Usez le style encapsulé dans un document/littéral. Dans ce cas, les messages possèdent toujours une portion unique, mais le référencement des pièces jointes peut être supprimé (comme décrit dans «Pièces jointes non référencées», à la page 55) ou les pièces jointes peuvent être typées swaRef (comme décrit dans «Pièces jointes référencées : éléments de type swaRef», à la page 45) dans ce cas.
2. Utilisé le style RPC/littéral. Dans ce cas, il n'existe aucune restriction sur la liaison WSDL en termes de nombre de portions associées au corps du message SOAP ; le message SOAP résultant possède toujours un enfant unique qui représente l'opération appelée, avec les portions de message correspondant aux enfants de cet élément.
3. Pour la liaison JAX-WS, la première portion de message ne doit pas être de type "hexBinary" ou "base64Binary" et toutes les autres portions doivent posséder l'un de ces deux types et seront alors associées comme pièces jointes.
4. Les autres cas ont le comportement décrit plus haut.

Remarque : Lorsqu'elle reçoit des messages SOAP de style document composite avec des pièces jointes référencées, la liaison JAX-WS s'attend à ce que chaque pièce jointe référencée soit représentée par un élément enfant de corps de message SOAP avec un attribut href dont la valeur identifie la pièce jointe d'après son ID contenu. La liaison JAX-WS envoie les pièces jointes référencées de tels messages de la même manière. Ce comportement n'est pas compatible avec WS-I Basic Profile. Le profil des pièces jointes WS-I définit un "codage des portions content-id" qui permet d'omettre l'élément enfant avec l'attribut href et donc de rendre de tels messages compatibles avec le profil de base. La liaison JAX-WS ne prend pas en charge l'envoi ou la réception de messages qui utilisent le codage de la portion content-id. Pour que vos messages soient compatibles, suivez l'approche 1 ou 2 dans la liste ci-dessus ou évitez d'utiliser des pièces jointes référencées pour de tels messages et utilisez à la place des pièces jointes non référencées ou typées swaRef.

Liaisons HTTP

La liaison HTTP permet de relier une architecture SOA à HTTP. Par conséquent, les applications HTTP existantes ou récemment développées peuvent être intégrées aux environnements d'architecture SOA (Service Oriented Architecture).

Le protocole HTTP (Hypertext Transfer Protocol) est largement utilisé pour le transfert d'informations sur le Web. Lorsque vous travaillez avec une application externe qui utilise le protocole HTTP, une liaison HTTP est nécessaire. La liaison HTTP gère la transformation des données entrantes en tant que message au format natif vers un objet métier dans une application SCA. La liaison HTTP peut également transformer les données sortantes en tant qu'objet métier au format natif attendu par l'application externe pour un message entrant.

Remarque : Si vous souhaitez interagir avec des clients et des services qui utilisent le protocole SOAP/HTTP des services Web, vous pouvez utiliser l'une des liaisons de service Web, qui offrent des fonctionnalités supplémentaires quant à la gestion des qualités de service standard des services Web.

La liste suivante répertorie certains scénarios courants pour l'utilisation de la liaison HTTP :

- Les services hébergés sur SCA peuvent appeler des applications HTTP en utilisant une importation HTTP.
- Les services hébergés sur SCA peuvent s'afficher eux-mêmes en tant qu'applications conformes à HTTP pour pouvoir être utilisés par des clients HTTP à l'aide d'une exportation HTTP.
- WebSphere Process Server et WebSphere Enterprise Service Bus peuvent communiquer entre eux via une infrastructure HTTP ; les utilisateurs peuvent donc gérer leurs communications en fonction de normes d'entreprise.
- WebSphere Process Server et WebSphere Enterprise Service Bus peuvent agir comme médiateurs des communications HTTP en tranformant et en routant les messages, ce qui améliore l'intégration des applications à l'aide d'un réseau HTTP.
- WebSphere Process Server et WebSphere Enterprise Service Bus peuvent être utilisés pour établir un pont entre HTTP et d'autres protocoles, comme les services Web SOAP/HTTP, les adaptateurs de ressources basés sur JCA (Java Connector Architecture), JMS, etc.

Pour plus d'informations sur la création de liaisons d'exportation et d'importation HTTP, consultez le centre de documentation de WebSphere Integration Developer. Voir les rubriques**Développement des applications d'intégration → Accès aux services externes avec HTTP**>.

Concepts associés

«Présentation des liaisons HTTP»
La liaison HTTP offre une connectivité aux applications hébergées sur HTTP. Elle sert d'intermédiaire aux communications entre les applications HTTP et permet aux applications existantes basées sur HTTP d'être appelées depuis un module.

«En-têtes HTTP», à la page 62
Les liaisons d'importation et d'exportation HTTP permettent d'effectuer la configuration des en-têtes HTTP et de leurs valeurs pour les messages sortants. L'importation HTTP utilise ces en-têtes pour les requêtes, tandis que l'exportation HTTP les exploite pour les réponses.

«Liaisons de données HTTP», à la page 67
Pour chaque mappage de données distinct entre un message SCA (Service Component Architecture) et un message de protocole HTTP, un gestionnaire de données ou une liaison de données HTTP doit être configuré. Les gestionnaires de données, qui fournissent une interface indépendante des liaisons pouvant être réutilisée avec différentes liaisons de transport, représentent l'approche recommandée ; les liaisons de données sont spécifiques à une liaison de transport particulière. Des classes de liaisons de données propres à HTTP sont fournies ; vous pouvez également créer des liaisons ou des gestionnaires de données personnalisés.

Présentation des liaisons HTTP

La liaison HTTP offre une connectivité aux applications hébergées sur HTTP. Elle sert d'intermédiaire aux communications entre les applications HTTP et permet aux applications existantes basées sur HTTP d'être appelées depuis un module.

Liaisons d'importation HTTP

La liaison d'importation HTTP offre une connectivité sortante depuis les applications SCA (Service Component Architecture) vers des applications ou un serveur HTTP.

L'importation appelle une adresse URL de noeud final HTTP. L'URl peut être spécifiée de l'une des trois façons suivantes :

- L'URL peut être définie de manière dynamique dans les en-têtes HTTP par l'intermdiaire d'une URL de substitution dynamique.
- L'URL peut être définie de manière dynamique dans l'élément d'adresse cible SMO.
- L'URL peut être spécifiée en tant que propriété de configuration sur l'importation.

Cet appel est toujours synchrone par nature.

Bien que les appels HTTP soient toujours de type demande-réponse, l'importation HTTP prend en charge à la fois les opérations unidirectionnelles et bidirectionnelles et ignore la réponse dans le cas d'une opération unidirectionnelle.

Liaisons d'exportation HTTP

La liaison d'exportation HTTP offre une connectivité entrante depuis les applications HTTP vers une application SCA.

Une adresse URL est définie sur l'exportation HTTP. Les applications HTTP qui veulent envoyer des messages de demande à l'exportation utilisent cette adresse URL pour appeler l'exportation.

L'exportation HTTP prend également en charge les commandes ping.

Liaisons HTTP en phase d'exécution

En phase d'exécution, une importation avec une liaison HTTP envoie une requête avec ou sans données dans le corps du message depuis l'application SCA vers le service Web externe. La requête est effectuée à partir de l'application SCA vers le service Web externe comme indiqué dans la figure 27.

Figure 27. Flux d'une requête depuis l'application SCA vers l'application Web

L'importation avec la liaison HTTP peut éventuellement recevoir des données en retour provenant de l'application Web dans une réponse à la requête.

Avec une exportation, la requête est faite par une application client vers un service Web comme indiqué dans la figure 28, à la page 62.

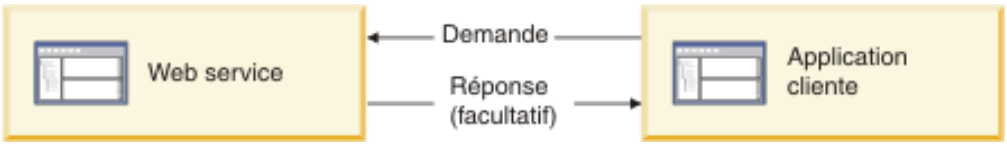


Figure 28. Flux d'une requête depuis le service Web vers une application client.

Le service Web est une application Web fonctionnant sur le serveur. L'exportation est implémentée dans cette application Web en tant que servlet de telle sorte que le client envoie sa requête à une adresse URL. Le servlet transmet la requête à l'application SCA dans l'environnement d'exécution.

L'exportation peut éventuellement envoyer des données à l'application client en réponse à la requête.

En-têtes HTTP

Les liaisons d'importation et d'exportation HTTP permettent d'effectuer la configuration des en-têtes HTTP et de leurs valeurs pour les messages sortants. L'importation HTTP utilise ces en-têtes pour les requêtes, tandis que l'exportation HTTP les exploite pour les réponses.

Les en-têtes et données de contrôle configurés de façon statique ont priorité sur les valeurs définies dynamiquement au moment de l'exécution. Toutefois, les valeurs de substitution dynamique d'adresse URL, de version et de méthode remplacement les valeurs statiques, qui sont dans les autres cas considérées comme valeurs par défaut.

La liaison prend en charge la nature dynamique de l'URL d'importation HTTP en déterminant la valeur des Méthode, Version et URL cible HTTP en phase d'exécution. Ces valeurs sont déterminées par l'extraction des valeurs de Référence de noeud final, URL de substitution dynamique, Version et Méthode.

- Pour Référence de noeud final, utilisez les API com.ibm.websphere.sca.addressing.EndpointReference ou définissez la zone /headers/SMOHeader/Target/address dans l'en-tête SMO.
- Pour URL de substitution dynamique, Version et Méthode, utilisez la section des paramètres de contrôle HTTP du message SCA (Service Component Architecture). Notez que l'URL de substitution dynamique est prioritaire par rapport à la référence de noeud final cible, mais que cette dernière s'appliquant entre les liaisons, elle est préférable et doit être utilisée chaque fois que possible.

Remarque : Voir Concepts associés pour plus d'informations sur l'appel dynamique et pour obtenir des informations spécifiques sur le format, la syntaxe et l'utilisation de l'URL.

Les données de contrôle et d'en-tête contenues dans les messages sortants sous les liaisons d'importation et d'exportation HTTP sont traitées dans l'ordre suivant :

1. Informations de contrôle et d'en-tête excluant les valeurs de méthode, de version et d'URL de substitution dynamique HTTP du message SCA (priorité la plus faible)
2. Les modifications effectuées au niveau de l'exportation/importation via la console d'administration
3. Les modifications effectuées au niveau de la méthode d'exportation ou d'importation via la console d'administration

4. Adresse cible spécifiée par l'intermédiaire de la référence de noeud final ou de l'en-tête SMO
5. URL de substitution dynamique, Version et Méthode du message SCA
6. Informations de contrôle et d'en-tête provenant du gestionnaire de données ou de la liaison de données (priorité supérieure)

L'importation et l'exportation HTTP ne rempliront les paramètres de contrôle et les en-têtes de direction sortants avec les données provenant du message entrant (HTTPExportRequest et HTTPImportResponse) que si la propagation de l'en-tête de protocole a la valeur `True`. Inversement, l'exportation et l'importation HTTP ne liront et ne traiteront les paramètres de contrôle et en-têtes sortants (HTTPExportResponse et HTTPImportRequest) que si la propagation de l'en-tête de protocole a la valeur `True`.

Remarque : Les modifications de la liaison de données ou du gestionnaire de données apportées aux en-têtes ou aux paramètres de contrôle dans la réponse d'importation ou la requête d'exportation n'altéreront pas les instructions de traitement du message au sein de la liaison d'imortation ou d'exportation et ne doivent être utilisées que pour diffuser les valeurs modifiées vers les composants SCA en aval.

Le service de contexte est chargé de la propagation du contexte (y compris les en-têtes de protocole comme l'en-tête HTTP et le contexte utilisateur comme l'ID de compte) tout au long d'un chemin d'appel SCA. Lors du développement dans WebSphere Integration Developer, vous pouvez contrôler la propagation du contexte par l'intermédiaire des propriétés d'importation et d'exportation. Pour plus de détails, reportez-vous aux informations relatives aux liaisons d'importation et d'exportation dans le centre de documentation de WebSphere Integration Developer.

Structures d'en-tête HTTP fournies et prise en charge

Le tableau 12 spécifie en détail les demandes de requête et de réponse d'importation HTTP et d'exportation HTTP.

Tableau 12. Informations d'en-tête HTTP fournies

Nom du contrôle	Demande d'importation HTTP	Réponse d'importation HTTP	Demande d'exportation HTTP	Réponse d'exportation HTTP
URL	Ignorée	Non définie	Lecture à partir du message de demande. **Remarque :** La chaîne de requête est également incluse dans le paramètre de contrôle de l'URL.	Ignorée
Version (valeurs possibles : 1.0, 1.1 - Par défaut : 1.1)	Ignorée	Non définie	Lecture à partir du message de demande	Ignorée

Tableau 12. Informations d'en-tête HTTP fournies (suite)

Nom du contrôle	Demande d'importation HTTP	Réponse d'importation HTTP	Demande d'exportation HTTP	Réponse d'exportation HTTP
Méthode	Ignorée	Non définie	Lecture à partir du message de demande	Ignorée
URL de substitution dynamique	Si elle est définie dans le gestionnaire de données ou la liaison de données, elle remplace l'URL d'importation HTTP. Inscrite sur le message dans la ligne de requête. **Remarque :** La chaîne de requête est également incluse dans le paramètre de contrôle de l'URL.	Non définie	Non définie	Ignorée
Version de substitution dynamique	Si elle est définie, elle se substitue à la version d'importation HTTP. Elle est inscrite sur le message dans la ligne de requête.	Non définie	Non définie	Ignorée
Méthode de substitution dynamique	Si elle est définie, elle se substitue à la méthode d'importation HTTP. Elle est inscrite sur le message dans la ligne de requête.	Non définie	Non définie	Ignorée

Tableau 12. Informations d'en-tête HTTP fournies (suite)

Nom du contrôle	Demande d'importation HTTP	Réponse d'importation HTTP	Demande d'exportation HTTP	Réponse d'exportation HTTP
Type de support (ce paramètre de contrôle transport une partie de la valeur de l'en-tête HTTP de type de contenu.)	Si elle existe, elle est inscrite dans le message en tant que partie de l'en-tête de type de contenu. **Remarque :** Cette valeur d'élément de contrôle doit être fournie par le gestionnaire de données ou la liaison de données.	Lecture à partir du message de réponse, en-tête Content-Type	Lecture à partir du message de demande, en-tête Content-Type	Si elle existe, elle est inscrite dans le message en tant que partie de l'en-tête Content-Type. **Remarque :** Cette valeur d'élément de contrôle doit être fournie par le gestionnaire de données ou la liaison de données.
Jeu de caractères (par défaut : UTF-8)	Si elle existe, elle est inscrite dans le message en tant que partie de l'en-tête de type de contenu. **Remarque :** Cette valeur d'élément de contrôle doit être fournie par la liaison de données.	Lecture à partir du message de réponse, en-tête Content-Type	Lecture à partir du message de demande, en-tête Content-Type	Pris en charge, inscrit dans le message en tant que partie de l'en-tête de type de contenu. **Remarque :** Cette valeur d'élément de contrôle doit être fournie par la liaison de données.
Codage de transfert (valeurs possibles : chunked, identity. Valeur par défaut: identity)	Si elle existe, elle est inscrite dans le message sous forme d'en-tête et contrôle le mode d'encodage appliqué lors de la transformation du message.	Lecture à partir du message de réponse	Lecture à partir du message de demande	Si elle existe, elle est inscrite dans le message sous forme d'en-tête et contrôle le mode d'encodage appliqué lors de la transformation du message.
Codage de contenu (valeurs possibles : gzip, x-gzip, deflate, identity. Valeur par défaut: identity)	Si elle existe, elle est inscrite dans le message sous forme d'en-tête et contrôle le mode d'encodage de la charge.	Lecture à partir du message de réponse	Lecture à partir du message de demande	Si elle existe, elle est inscrite dans le message sous forme d'en-tête et contrôle le mode d'encodage de la charge.
Longueur de contenu	Ignorée	Lecture à partir du message de réponse	Lecture à partir du message de demande	Ignorée
StatusCode (par défaut : 200)	Non prise en charge.	Lecture à partir du message de réponse	Non prise en charge.	Si elle existe, elle est inscrite dans le message dans la ligne de réponse

Tableau 12. Informations d'en-tête HTTP fournies (suite)

Nom du contrôle	Demande d'importation HTTP	Réponse d'importation HTTP	Demande d'exportation HTTP	Réponse d'exportation HTTP
ReasonPhrase (par défaut : OK)	Non prise en charge.	Lecture à partir du message de réponse	Non prise en charge.	Valeur de contrôle ignorée. La valeur contenue sur la ligne de réponse du message est générée à partir de StatusCode.
Authentification (propriétés multiples)	Si elle existe, elle est utilisée pour établir l'en-tête d'authentification de base. **Remarque :** La valeur de cet en-tête est uniquement encodée dans le protocole HTTP. Dans l'architecture SCA, cette donnée est décodée et transmise sous forme de texte en clair.	Non applicable	Lecture à partir de l'en-tête d'authentification de base du message de demande. La présence de cet en-tête n'indique pas que l'utilisateur a été authentifié. Il convient de contrôler l'authentification via la configuration du servlet. **Remarque :** La valeur de cet en-tête est uniquement encodée dans le protocole HTTP. Dans l'architecture SCA, cette donnée est décodée et transmise sous forme de texte en clair.	Non applicable
Proxy (contient des propriétés multiples : Host, Port, Authentication)	Si elle existe, elle permet d'établir la connexion via le serveur Proxy.	Non applicable	Non applicable	Non applicable
SSL (contient des propriétés multiples : Keystore, Keystore Password, Trustore, Trustore Password, ClientAuth)	Si elle est remplie et que L'URL de destination est HTTPS, elle est utilisée pour établir une connexion via SSL.	Non applicable	Non applicable	Non applicable

Liaisons de données HTTP

Pour chaque mappage de données distinct entre un message SCA (Service Component Architecture) et un message de protocole HTTP, un gestionnaire de données ou une liaison de données HTTP doit être configuré. Les gestionnaires de données, qui fournissent une interface indépendante des liaisons pouvant être réutilisée avec différentes liaisons de transport, représentent l'approche recommandée ; les liaisons de données sont spécifiques à une liaison de transport particulière. Des classes de liaisons de données propres à HTTP sont fournies ; vous pouvez également créer des liaisons ou des gestionnaires de données personnalisés.

Remarque : Les trois classes de liaison de données HTTP (HTTPStreamDataBindingSOAP, HTTPStreamDataBindingXML et HTTPServiceGatewayDataBinding) sont dépréciées à partir de WebSphere Process Server, version 7.0. Au lieu d'utiliser les liaisons de données décrites dans cette rubrique, prenez en compte les gestionnaires de données suivants :

- Utilisez SOAPDataHandler au lieu de HTTPStreamDataBindingSOAP.
- Utilisez UTF8XMLDataHandler au lieu de HTTPStreamDataBindingXML
- Utilisez GatewayTextDataHandler au lieu de HTTPServiceGatewayDataBinding

Les liaisons de données suivantes sont disponibles pour les importations et les exportations HTTP : liaison de données binaire, liaison de données XML et liaison de données SOAP. Une liaison de données de réponse n'est pas nécessaire pour les opérations unidirectionnelles. Une liaison de données est représentée par le nom d'une classe Java dont les instances permettent à la fois les conversions de HTTP vers ServiceDataObject et vice versa. Un sélecteur de fonction doit être utilisé lors d'une exportation pour déterminer, conjointement avec les liaisons de méthodes, la liaison de données utilisée et l'opération appelée. Les liaisons de données fournies sont les suivantes :

- Les liaisons de données binaires qui traitent le corps des messages comme des données binaires non structurées. L'implémentation du schéma XSD de liaisons de données binaires se présente comme suit :

```
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:tns="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="HTTPBaseBody">
    <xsd:sequence/>
  </xsd:complexType>

  <xsd:complexType name="HTTPBytesBody">
    <xsd:complexContent>
      <xsd:extension base="tns:HTTPBaseBody">
        <xsd:sequence>
          <xsd:element name="value" type="xsd:hexBinary"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

- Les liaisons de données XML qui prennent en charge le corps des messages en tant que données XML. L'implémentation des liaisons de données XML est similaire à celle des liaisons de données XML JMS et n'impose aucune restriction quant au schéma d'interface.
- Les liaisons de données SOAP qui prennent en charge le corps des messages en tant que données SOAP. L'implémentation des liaisons de données SOAP n'impose aucune restriction quant au schéma d'interface.

Implémentation des liaisons de données HTTP personnalisées

Cette section indique comment implémenter une liaison de données HTTP personnalisée.

Remarque : L'approche recommandée consiste à implémenter un gestionnaire de données personnalisé car il peut être réutilisé avec différentes liaisons de transport.

HTTPStreamDataBinding est la principale interface de gestion des messages HTTP personnalisés. Cette interface est conçue pour permettre le traitement de charges utiles importantes. Toutefois, pour que ce type d'implémentation fonctionne, cette liaison de données doit renvoyer les informations de contrôle et les en-têtes avant que le message ne soit inséré dans le flux.

Les méthodes et leur ordre d'exécution (indiqué ci-dessous) doivent être implémentés par la liaison de données personnalisée.

Pour personnaliser une liaison de données, créez une classe implémentant HTTPStreamDataBinding. La liaison de données doit comporter les quatre propriétés privées suivantes :

- private DataObject pDataObject
- private HTTPControl pCtrl
- private HTTPHeaders pHeaders
- private yourNativeDataType nativeData

La liaison HTTP appellera la liaison de données personnalisée dans l'ordre suivant :

- Traitement sortant (objet de données vers format natif) :
  1. setDataObject(...)
  2. setHeaders(...)
  3. setControlParameters(...)
  4. setBusinessException(...)
  5. convertToNativeData()
  6. getControlParameters()
  7. getHeaders()
  8. write(...)
- Traitement entrant (format natif vers objet de données) :
  1. setControlParameters(...)
  2. setHeaders(...)
  3. convertFromNativeData(...)
  4. isBusinessException()
  5. getDataObject()
  6. getControlParameters()
  7. getHeaders()

Vous devez appeler setDataObject(...) dans convertFromNativeData(...) pour définir la valeur de dataObject, qui est convertie du format de données natif en propriété privée "pDataObject".

```
public void setDataObject(DataObject dataObject)
   throws DataBindingException {
  pDataObject = dataObject;
```

```
}
public void setControlParameters(HTTPControl arg0) {
  this.pCtrl = arg0;
}

public void setHeaders(HTTPHeaders arg0) {
  this.pHeaders = arg0;
}
/*
* Ajouter l'en-tête http "IsBusinessException" dans pHeaders.
* Deux étapes :
* 1. Supprimer d'abord tous les en-têtes portant le nom IsBusinessException
(insensible à la casse).
*   Cette opération garantit qu'un seul en-tête est présent.
* 2. Ajouter le nouvel en-tête "IsBusinessException"
*/
public void setBusinessException(boolean isBusinessException) {
  //Supprimer d'abord tous les en-têtes portant le nom IsBusinessException
(insensible à la casse).
  //Cette opération garantit qu'un seul en-tête est présent.
  //Ajouter le nouvel en-tête "IsBusinessException", exemple de code :
  HTTPHeader header=HeadersFactory.eINSTANCE.createHTTPHeader();
  header.setName("IsBusinessException");
  header.setValue(Boolean.toString(isBusinessException));
  this.pHeaders.getHeader().add(header);

}
public HTTPControl getControlParameters() {
  return pCtrl;
}
public HTTPHeaders getHeaders() {
  return pHeaders;
}
public DataObject getDataObject() throws DataBindingException {
  return pDataObject;
}
/*
* Extraire l'en-tête "IsBusinessException" de pHeaders, renvoyer sa valeur booléenne
*/
public boolean isBusinessException() {
  String headerValue = getHeaderValue(pHeaders,"IsBusinessException");
  boolean result=Boolean.parseBoolean(headerValue);
  return result;
}
public void convertToNativeData() throws DataBindingException {
  DataObject dataObject = getDataObject();
  this.nativeData=realConvertWorkFromSDOToNativeData(dataObject);
}
public void convertFromNativeData(HTTPInputStream arg0){
  //Méthode développée par le client pour
  //lire les données de HTTPInputStream
  //et les convertir en objet de données (DataObject)
  DataObject dataobject=realConvertWorkFromNativeDataToSDO(arg0);
  setDataObject(dataobject);
}
public void write(HTTPOutputStream output) throws IOException {
  if (nativeData != null)
  output.write(nativeData);
}
```

Liaisons EJB

Les liaisons d'importation EJB (Enterprise JavaBeans permettent aux composants SCA d'appeler les services fournis par la logique métier de Java EE exécutée sur un serveur Java EE. Les liaisons d'exportation EJB permettent aux composants d'être

affichés comme des EJB (Enterprise JavaBeans) pour que la logique métier de Java EE puisse appeler les composants SCA qui ne seraient autrement pas disponibles.

Concepts associés

«Liaisons d'importation EJB»
Les liaisons d'importation EJB permettent à un module SCA d'appeler des implémentations EJB en indiquant la manière dont le module utilisateur est lié à l'EJB externe. L'importation de services à partir d'une implémentation EJB externe permet aux utilisateurs de relier leur logique métier à l'environnement WebSphere Process Server et de participer à un processus métier.

«Liaisons d'exportation EJB», à la page 71
Les applications Java EE externes peuvent appeler un composant SCA par l'intermédiaire d'une liaison d'exportation EJB. L'utilisation d'une exportation EJB permet d'exposer des composants SCA pour que les applications Java EE externes puissent appeler ces composants à l'aide du modèle de programmation des EJB.

«Propriétés des liaisons EJB», à la page 73
Les liaisons d'importation EJB utilisent leurs noms JNDI configurés pour déterminer le niveau du modèle de programmation d'EJB et le type d'appel (local ou éloigné). Les liaisons d'importation et d'exportation EJB utilisent le gestionnaire de données JAX-WS pour la transformation des données. La liaison d'importation EJB utilise un sélecteur de fonction d'importation EJB et un sélecteur d'erreur EJB, tandis que la liaison d'exportation EJB utilise un sélecteur de fonction d'exportation EJB.

Liaisons d'importation EJB

Les liaisons d'importation EJB permettent à un module SCA d'appeler des implémentations EJB en indiquant la manière dont le module utilisateur est lié à l'EJB externe. L'importation de services à partir d'une implémentation EJB externe permet aux utilisateurs de relier leur logique métier à l'environnement WebSphere Process Server et de participer à un processus métier.

Vous pouvez utiliser WebSphere Integration Developer pour créer des liaisons d'importation EJB. Vous pouvez utiliser l'une des procédures suivantes pour générer les liaisons :

- Création d'une importation EJB à l'aide de l'assistant de service externe

  Vous pouvez utiliser l'assistant de service externe de WebSphere Integration Developer pour générer une importation EJB à partir d'une implémentation existante. L'assistant de service externe crée des services en fonction des critères que vous fournissez. Il génère ensuite des objets métier, des interfaces et des fichiers d'importation à partir des services détectés.
- Création d'une importation EJB à l'aide de l'éditeur d'assemblage

  Vous pouvez créer une importation EJB dans un diagramme d'assemblage à l'aide de l'éditeur d'assemblage de WebSphere Integration Developer. A partir de la palette, vous pouvez utiliser une importation ou utiliser une classe Java pour créer la liaison EJB.

L'importation générée contient des liaisons de données qui établissent la connexion Java-WSDL, ce qui évite le recours à un composant de passerelle Java. Vous pouvez connecter directement un composant avec une référence WSDL (Web Services Description Language) à l'importation EJB qui communique avec un service EJB à l'aide d'une interface Java.

L'importation EJB peut interagir avec la logique métier Java EE à l'aide du modèle de programmation EJB 2.1 ou du modèle de programmation EJB 3.0.

L'appel de la logique métier Java EE peut être local (uniquement pour EJB 3.0) ou éloigné.

- L'appel local est utilisé lorsque vous souhaitez appeler la logique métier Java EE qui se trouve sur le même serveur que l'importation.

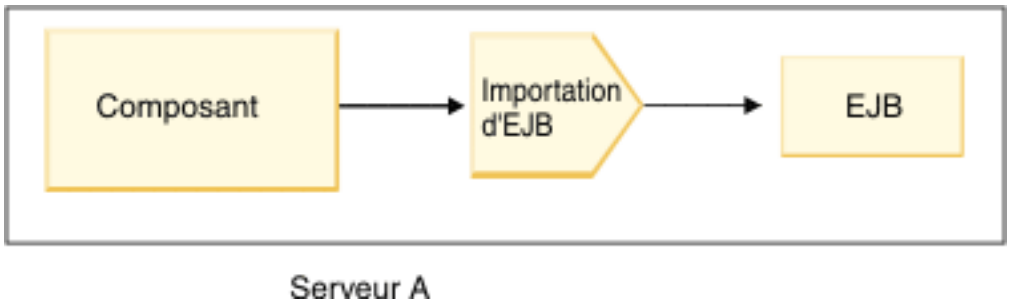


Figure 29. Appel en local d'un EJB (EJB 3.0 uniquement)

- L'appel éloigné est utilisé lorsque vous souhaitez appeler la logique métier Java EE qui ne se trouve pas sur le même serveur que l'importation.

  Par exemple, dans la figure ci-après, une importation EJB utilise RMI/IIOP (Remote Method Invocation over Internet InterORB Protocol) pour appeler une méthode EJB sur un autre serveur.

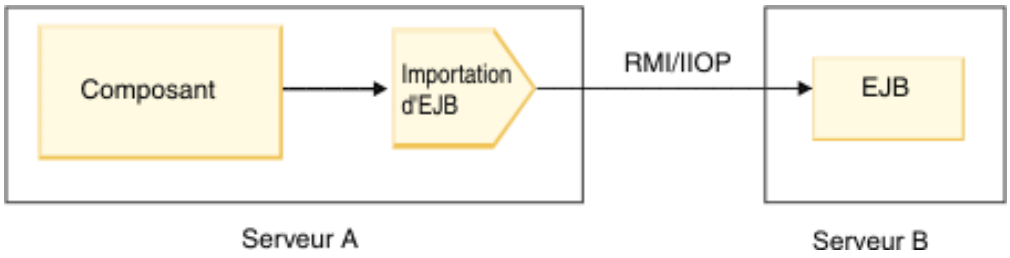


Figure 30. Appel éloigné d'un EJB

Lorsqu'il configure la liaison EJB, WebSphere Integration Developer utilise le nom JNDI pour déterminer le niveau du modèle de programmation d'EJB et le type d'appel (local ou éloigné).

Les liaisons d'importation EJB contiennent les principaux composants suivants :

- Gestionnaire de données JAX-WS
- Sélecteur d'erreurs EJB
- Sélecteur de fonction d'importation EJB

Si votre scénario utilisateur ne repose pas sur le mappage JAX-WS, vous risquez d'avoir besoin d'un gestionnaire de données, d'un sélecteur de fonction et d'un sélecteur d'erreur personnalisés pour accomplir les tâches habituellement dévolues aux composants des liaisons d'importation EJB. Ces tâches incluent le mappage normalement effectué par l'algorithme de mappage personnalisé.

Liaisons d'exportation EJB

Les applications Java EE externes peuvent appeler un composant SCA par l'intermédiaire d'une liaison d'exportation EJB. L'utilisation d'une exportation EJB permet d'exposer des composants SCA pour que les applications Java EE externes puissent appeler ces composants à l'aide du modèle de programmation des EJB.

Remarque : L'exportation EJB est un bean sans état.

Vous pouvez utiliser WebSphere Integration Developer pour créer des liaisons EJB. Vous pouvez utiliser l'une des procédures suivantes pour générer les liaisons :

- Création de liaisons d'exportation EJB à l'aide de l'assistant de service externe

  Vous pouvez utiliser l'assistant de service externe de WebSphere Integration Developer pour générer un service d'exportation EJB à partir d'une implémentation existante. L'assistant de service externe crée des services en fonction des critères que vous fournissez. Il génère ensuite des objets métier, des interfaces et des fichiers d'exportation à partir des services détectés.
- Création de liaisons d'exportation EJB à l'aide de l'éditeur d'assemblage.

  Vous pouvez créer une exportation EJB à l'aide de l'éditeur d'assemblage de WebSphere Integration Developer.

Vous pouvez générer la liaison à partir d'un composant SCA existant ou générer une exportation avec une liaison EJB pour une interface Java.

- Lorsque vous générez une exportation pour un composant SCA existant qui possède une interface WSDL, une interface Java est affectée à l'exportation.
- Lorsque vous générez une exportation pour une interface Java, vous pouvez sélectionner un WSDL ou une interface Java pour l'exportation.

  Remarque : Une interface Java utilisée pour créer une exportation EJB est soumise aux limitations suivantes en ce qui concerne les objets (exceptions et paramètres en entrée et en sortie) transmis comme paramètres sur un appel éloigné :
  - Ils doivent être de type concret (au lieu d'une interface ou d'un type abstrait).
  - Ils doivent respecter la spécification Enterprise JavaBean. Ils doivent être sérialisables et contenir le constructeur no-argument par défaut et toutes les propriétés doivent être accessibles par l'intermédiaire de méthodes getter et setter.

    Pour des informations sur la spécification Enterprise JavaBean, reportez-vous au site Web de Sun Microsystems, Inc., à l'adresse http://java.sun.com.

  En outre, l'exception doit correspondre à une exception vérifiée, héritée de java.lang.Exception et doit être unique (la génération de plusieurs exceptions de type vérifiée n'est pas prise en charge).

  Notez par ailleurs que l'interface métier d'un bean enterprise Java est une interface Java standard et ne doit pas étendre javax.ejb.EJBObject ou javax.ejb.EJBLocalObject. Les méthodes de l'interface métier ne doivent pas générer d'exception java.rmi.Remote.Exception.

Les liaisons d'exportation EJB peuvent interagir avec la logique métier Java EE à l'aide du modèle de programmation EJB 2.1 ou du modèle de programmation EJB 3.0.

L'appel peut être local (uniquement pour EJB 3.0) ou éloigné.

- L'appel local est utilisé lorsque la logique métier Java EE appelle un composant SCA qui se trouve sur le même serveur que l'exportation.
- L'appel éloigné est utilisé lorsque la logique métier Java EE ne se trouve pas sur le même serveur que l'exportation.

Par exemple, dans la figure ci-après, un EJB utilise RMI/IIOP pour appeler un composant SCA sur un autre serveur.

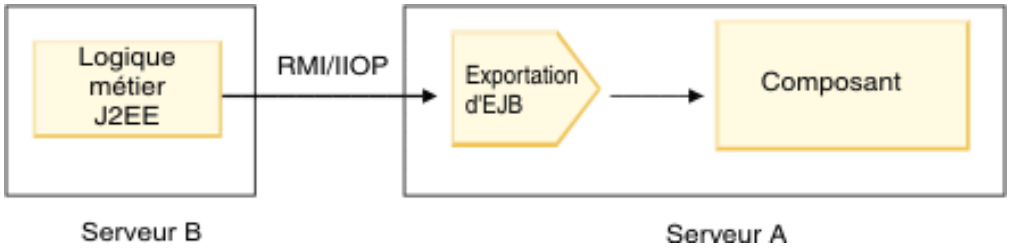


Figure 31. Appel éloigné d'un client vers un composant SCA par l'intermédiaire d'une exportation EJB

Lorsqu'il configure la liaison EJB, WebSphere Integration Developer utilise le nom JNDI pour déterminer le niveau du modèle de programmation d'EJB et le type d'appel (local ou éloigné).

Les liaisons d'exportation EJB contiennent les principaux composants suivants :

- Gestionnaire de données JAX-WS
- Sélecteur de fonction d'exportation EJB

Si votre scénario utilisateur ne repose pas sur le mappage JAX-WS, vous risquez d'avoir besoin d'un gestionnaire de données et d'un sélecteur de fonction personnalisés pour accomplir les tâches habituellement dévolues aux composants des liaisons d'exportation EJB. Ces tâches incluent le mappage normalement effectué par l'algorithme de mappage personnalisé.

Propriétés des liaisons EJB

Les liaisons d'importation EJB utilisent leurs noms JNDI configurés pour déterminer le niveau du modèle de programmation d'EJB et le type d'appel (local ou éloigné). Les liaisons d'importation et d'exportation EJB utilisent le gestionnaire de données JAX-WS pour la transformation des données. La liaison d'importation EJB utilise un sélecteur de fonction d'importation EJB et un sélecteur d'erreur EJB, tandis que la liaison d'exportation EJB utilise un sélecteur de fonction d'exportation EJB.

Concepts associés

«Noms JNDI et liaisons d'importation EJB»
Lorsqu'il configure la liaison EJB lors d'une importation, WebSphere Integration Developer utilise le nom JNDI pour déterminer le niveau du modèle de programmation d'EJB et le type d'appel (local ou éloigné).

«Gestionnaire de données JAX-WS», à la page 75
La liaison d'importation EJB (Enterprise JavaBeans) utilise le gestionnaire de données JAX-WS pour transformer les objets métier de demande en paramètres d'objet Java et la valeur renvoyée par l'objet Java en objet métier de réponse. La liaison d'exportation EJB utilise le gestionnaire de données JAX-WS pour transformer les EJB de demande en objets métier de demande et l'objet métier de réponse en valeur renvoyée.

«Sélecteur d'erreur EJB», à la page 77
Le sélecteur d'erreur EJB détermine si un appel EJB a engendré une erreur, une exception d'exécution ou une réponse correcte.

«Sélecteur de fonction EJB», à la page 77
Les liaisons EJB utilisent un sélecteur de fonction d'importation (pour le traitement sortant) un sélecteur de fonction d'exportation (pour le traitement entrant) afin de déterminer la méthode EJB à appeler.

Noms JNDI et liaisons d'importation EJB :

Lorsqu'il configure la liaison EJB lors d'une importation, WebSphere Integration Developer utilise le nom JNDI pour déterminer le niveau du modèle de programmation d'EJB et le type d'appel (local ou éloigné).

Si aucun nom JNDI n'est spécifié, la liaison d'interface EJB est utilisée. Les noms par défaut créés varient suivant que vous appeliez un bean Java EJB 2.1 ou EJB 3.0.

Remarque : Pour des informations plus détaillées sur les conventions d'affectation de nom, reportez-vous au centre de documentation de WebSphere Application Server : Présentation des liaisons d'application EJB 3.0.

- Bean Java EJB 2.1

  Le nom JNDI par défaut présélectionné par WebSphere Integration Developer correspond à la liaison EJB 2.1 par défaut, qui prend la forme `ejb/` plus l'interface home, séparés par des barres obliques.

  Par exemple, pour l'interface home d'un bean Java EJB 2.1 de com.mycompany.myremotebusinesshome, la liaison par défaut est :

  ```
  ejb/com/mycompany/myremotebusinesshome
  ```

  Pour EJB 2.1, seul l'appel EJB éloigné est pris en charge.
- Bean Java EJB 3.0

  Le nom JNDI par défaut présélectionné par WebSphere Integration Developer pour le JNDI local correspond au nom de classe complet de l'interface locale précédé d'`ejblocal:`. Par exemple, pour l'interface complète de l'interface com.mycompany.mylocalbusiness locale, le JNDI EJB 3.0 présélectionné est :

  ```
  ejblocal:com.mycompany.mylocalbusiness
  ```

  Pour l'interface com.mycompany.myremotebusiness éloignée, le JNDI EJB 3.0 présélectionné correspond à l'interface complète :

  ```
  com.mycompany.myremotebusiness
  ```

  Les liaisons d'application par défaut EJB 3.0 sont décrites dans l'emplacement suivant : Présentation des liaisons d'application EJB 3.0.

WebSphere Integration Developer utilise le nom "abrégé" comme emplacement JNDI par défaut des EJB à l'aide du modèle de programmation de la version 3.0.

Remarque : Si la référence JNDI déployée de l'EJB cible est différente de l'emplacement de la liaison JNDI par défaut car un mappage personnalisé a été utilisé ou configuré, le nom JNDI cible doit être correctement spécifié. Vous pouvez spécifier le nom dans WebSphere Integration Developer avant le déploiement ou, pour la liaison d'importation, vous pouvez modifier le nom dans la console d'administration (après le déploiement), pour qu'il corresponde au nom JNDI de l'EJB cible.

Pour plus d'informations sur la création de liaisons EJB, reportez-vous à la section relative à l'utilisation des liaisons EJB dans le WebSphere Integration Developer centre de documentation.

Gestionnaire de données JAX-WS :

La liaison d'importation EJB (Enterprise JavaBeans) utilise le gestionnaire de données JAX-WS pour transformer les objets métier de demande en paramètres d'objet Java et la valeur renvoyée par l'objet Java en objet métier de réponse. La liaison d'exportation EJB utilise le gestionnaire de données JAX-WS pour transformer les EJB de demande en objets métier de demande et l'objet métier de réponse en valeur renvoyée.

Ce gestionnaire de données mappe les données de l'interface spécifiée par SCA à l'interface Java de l'EJB cible (et vice versa) à l'aide de la spécification JAX-WS (Java API for XML Web Services) et de la spécification JAXB (Java Architecture for XML Binding).

Remarque : Le support actuel est limité aux spécifications JAX-WS 2.1.1 et JAXB 2.1.3.

Le gestionnaire de données spécifié au niveau de la liaison EJB est utilisé pour traiter les demandes, les réponses, les incidents et les exceptions d'exécution.

Remarque : Pour les erreurs, un gestionnaire de données propre peut être spécifié pour chaque erreur en indiquant la propriété de configuration faultBindingType. Cette dernière remplace la valeur spécifiée au niveau de la liaison EJB.

Le gestionnaire de données JAX-WS est utilisé par défaut lorsque la liaison EJB dispose d'une interface WSDL. Ce gestionnaire de données ne peut pas être utilisé pour transformer un message SOAP représentant un appel JAX-WS en objet de données.

La liaison d'importation EJB utilise un gestionnaire de données pour transformer un objet de données en matrice d'objets Java (Object[]). Lors des communications sortantes, le traitement suivant est effectué :

1. La liaison EJB définit le type et l'élément attendus ainsi que le nom de la méthode ciblée dans BindingContext, afin qu'ils soient identiques à ceux spécifiés dans le fichier WSDL.
2. La liaison EJB appelle la méthode de transformation pour l'objet de données nécessitant une transformation des données.
3. Le gestionnaire de données renvoie un tableau Object[] représentant les paramètres de la méthode (listés dans l'ordre suivant lequel ils sont définis dans la méthode).

4. La liaison EJB utilise le tableau Object[] pour appeler la méthode sur l'interface EJB cible.

La liaison prépare également un tableau Object[] pour traiter la réponse à partir de l'appel EJB.

- Le premier élément dans l'objet[] correspond à la valeur renvoyée par l'appel de méthode Java.
- Les valeurs suivantes représentent les paramètres d'entrée de la méthode.

Ces dernières sont nécessaires pour prendre en charge les paramètres de type entrée/sortie et sortie.

Pour les paramètres de type sortie, les valeurs doivent être renvoyées dans l'objet de données de réponse.

Le gestionnaire de données traite et transforme les valeurs rencontrées dans l'objet[], puis renvoie une réponse à l'objet de données.

Le gestionnaire de données prend en charge xs:AnyType, xs:AnySimpleType, and xs:Any, ainsi que d'autres types de données XSD. Pour activer la prise en charge de xs:Any, utilisez `@XmlAnyElement (lax=true)` pour la propriété JavaBean dans le code Java, comme illustré dans l'exemple suivant :

```
public class TestType {
 private Object[] object;

 @XmlAnyElement (lax=true)
 public Object[] getObject() {
  return object;
 )

 public void setObject (Object[] object) {
  this.object=object;
 )
)
```

Cela permet de transformer l'objet de propriété dans TestType en champ xs:any. La valeur de classe Java utilisée dans le champ xs:any doit inclure l'annotation `@XmlAnyElement`. Par exemple, si Adresse est la classe Java utilisée pour remplir la matrice d'objets, elle doit comporter l'annotation `@XmlRootElement`.

Remarque : Pour personnaliser le mappage du type XSD aux types Java définis par la spécification JAX-WS, modifiez les annotations JAXB selon les besoins de votre entreprise. Le gestionnaire de données JAX-WS prend en charge xs:any, xs:anyType et xs:anySimpleType.

Les restrictions suivantes s'appliquent au gestionnaire de données JAX-WS :

- Le gestionnaire de données ne prend pas en charge l'annotation `@WebParam` de l'attribut d'en-tête.
- L'espace de noms pour les fichiers de schéma des objet métiers (fichiers XSD) n'inclut pas le mappage par défaut du nom de package Java. L'annotation `@XMLSchema` dans package-info.java ne fonctionne pas non plus. La seule manière de créer un fichier XSD avec un espace de nom consiste à utiliser les annotations `@XmlType` et `@XmlRootElement`. `@XmlRootElement` définit le nom d'espace cible pour l'élément global dans les types de Javabean.
- L'assistant d'importation EJB ne crée pas de fichiers XSD pour les classes non associées. La version 2.0 ne prend pas en charge l'annotation `@XmlSeeAlso`. Par

conséquent, si la classe enfant n'est pas directement référencée par la classe parente, aucun fichier XSD n'est créé. La solution à ce problème consiste à exécuter SchemaGen pour ces classes enfants.

SchemaGen est un utilitaire de ligne de commande (situé dans le répertoire *Rép_base_installation_WPS*/bin) fourni pour créer des fichiers XSD pour un JavaBean donné. Ces fichiers XSD doivent être manuellement copiés vers le module pour que la solution fonctionne.

Sélecteur d'erreur EJB :

Le sélecteur d'erreur EJB détermine si un appel EJB a engendré une erreur, une exception d'exécution ou une réponse correcte.

Si une erreur est détectée, le sélecteur d'erreurs EJB renvoie le nom de l'erreur native à l'environnement d'exécution de la liaison afin que le gestionnaire de données JAX-WS puisse convertir l'objet exception en objet métier d'erreur.

En cas de réponse positive (pas d'erreur), la liaison d'importation EJB assemble une matrice d'objets Java (Object[]) pour renvoyer les valeurs.

- Le premier élément dans l'objet[] correspond à la valeur renvoyée par l'appel de méthode Java.
- Les valeurs suivantes représentent les paramètres d'entrée de la méthode.

Ces dernières sont nécessaires pour prendre en charge les paramètres de type entrée/sortie et sortie.

Pour les scénarios d'exception, la liaison assemble un objet[] et le premier élément représente l'exception générée par la méthode.

Le sélecteur d'erreurs peut renvoyer l'une des valeurs suivantes :

Tableau 13. Valeurs renvoyées

Type	Valeur renvoyée	Description
Erreur	`ResponseType.FAULT`	Renvoyée si la matrice Object[] transmise contient un objet d'exception.
Exception d'exécution	`ResponseType.RUNTIME`	Renvoyée si l'objet d'exception ne correspond à aucun des types d'exception déclarés sur la méthode.
Réponse normale	`ResponseType.RESPONSE`	Renvoyée dans tous les autres cas.

Si le sélecteur d'erreurs renvoie la valeur `ResponseType.FAULT`, le nom d'erreur natif est renvoyé. Ce nom est utilisé par la liaison pour déterminer le nom d'erreur WSDL correspondant à partir du modèle et appeler le gestionnaire de données d'erreur approprié.

Sélecteur de fonction EJB :

Les liaisons EJB utilisent un sélecteur de fonction d'importation (pour le traitement sortant) un sélecteur de fonction d'exportation (pour le traitement entrant) afin de déterminer la méthode EJB à appeler.

Sélecteur de fonction d'importation

Pour le traitement sortant, le sélecteur de fonction d'importation détermine le type de méthode EJB en fonction du nom de l'opération appelée par le composant SCA connecté à l'importation EJB. Le sélecteur de fonction recherche l'annotation @WebMethod sur la classe Java annotée par JAS-WS et générée par WebSphere Integration Developer pour déterminer le nom de l'opération cible associée.

- Si l'annotation @WebMethod est présente, le sélecteur de fonction utilise l'annotation @WebMethod pour déterminer le mappage correct de la méthode Java pour la méthode WSDL.
- Si l'annotation @WebMethod est manquante, le sélecteur de fonction suppose que le nom de la méthode Java est identique au nom de l'opération appelée.

Remarque : Ce sélecteur de fonction n'est valide que pour une interface de type WSDL sur une importation EJB et non pour une interface de type Java sur une importation EJB.

Le sélecteur de fonction renvoie un objet java.lang.reflect.Method qui représente la méthode de l'interface EJB.

Le sélecteur de fonction utilise une matrice d'objets Java (Object[]) pour conserver la réponse de la méthode cible. Le premier élément de la matrice Object[] est une méthode Java possédant le nom du WSDL et le second correspond à l'objet métier en entrée.

Sélecteur de fonction d'exportation

Pour le traitement entrant, le sélecteur de fonction d'exportation détermine la méthode cible à appeler à partir de la méthode Java.

Le sélecteur de fonction d'exportation mappe le nom de l'opération Java appelée par le client EJB au nom de l'opération dans l'interface du composant cible. Le nom de la méthode est renvoyé sous forme de chaîne et résolu par l'environnement d'exécution de SCA en fonction du type d'interface du composant cible.

Liaisons EIS

Les liaisons EIS (Enterprise Information System) assurent la connectivité entre les composants SCA et un système d'information d'entreprise externe. Cette communication est accomplie à l'aide des importations et exportations EIS qui prennent en charge les adaptateurs de ressources JCA 1.5 et les adaptateurs Websphere.

Vos composants SCA peuvent exiger le transfert de données vers ou depuis un EIS externe. Lorsque vous créez un module SCA exigeant une telle connectivité, vous incluez (outre le composant SCA) une importation ou exportation avec une liaisons EIS pour communiquer avec un EIS externe spécifique.

Les adaptateurs de ressources dans WebSphere Integration Developer sont utilisés dans le contexte d'une importation ou d'une exportation. Vous développez une importation ou une exportation à l'aide de l'assistant de service externe puis, lors du développement vous incluez l'adaptateur de ressources. Une importation EIS permettant à votre application d'appeler un service sur un système EIS ou une exportation EIS permettant à une application sur un système EIS d'appeler un service développé dans WebSphere Integration Developer, sont créées avec un

adaptateur de ressources particulier. Par exemple, vous pourriez créer une importation avec un adaptateur JD Edwards pour appeler un service sur le système JD Edwards.

Lorsque vous utilisez un assistant de service externe, les informations de liaison EIS sont créées pour vous. Vous pouvez également utiliser un autre outil, l'éditeur d'assemblage, pour ajouter ou modifier des informations de liaison. Pour plus d'informations, voir le centre de documentation de WebSphere Integration Developer.

Une fois que le module SCA contenant la liaison EIS est déployé sur le serveur, vous pouvez utiliser la console d'administration pour afficher les informations relatives à la liaison ou pour la configurer.

Concepts associés

«Présentation des liaisons EIS»
Lorsqu'elle est utilisée avec un adaptateur de ressources JCA, la liaison EIS (Enterprise Information System) permet d'accéder à des services sur un système d'information d'entreprise ou de rendre vos services disponibles auprès de l'EIS.

«Propriétés dynamiques des spécifications d'interaction et de connexion JCA», à la page 83
La liaison EIS peut accepter des données en entrée pour lesspécifications InteractionSpec et ConnectionSpec spécifiées, en utilisant un objet de données enfant bien défini qui accompagne la charge. Ceci permet des interactions demande-réponse dynamiques avec un adaptateur de ressources par le biais de InteractionSpec et l'authentification des composants par le biais de ConnectionSpec.

«Clients externes et liaisons EIS», à la page 84
Le serveur peut envoyer des messages à, ou recevoir des messages de clients externes par le biais de liaisons EIS.

Référence associée

«Principales fonctionnalités des liaisons EIS», à la page 80
Une importation EIS est une importation SCA (Service Component Architecture) qui permet aux composants du module SCA d'utiliser les applications EIS définies hors du module SCA. Une importation EIS permet de transférer des données d'un composant SCA vers un système EIS externe. Une exportation EIS permet de transférer des données d'un système EIS externe vers un module SCA.

Présentation des liaisons EIS

Lorsqu'elle est utilisée avec un adaptateur de ressources JCA, la liaison EIS (Enterprise Information System) permet d'accéder à des services sur un système d'information d'entreprise ou de rendre vos services disponibles auprès de l'EIS.

L'exemple suivant illustre la manière dont un module SCA appelé ContactSyncModule synchronise les informations de contact entre un système Siebel et un système SAP.

1. Le composant SCA appelé ContactSync écoute (via une exportation d'application EIS appelée Siebel Contact) les modifications apportées aux contacts Siebel.
2. Le composant ContactSync lui-même utilise une application SAP (via une importation d'application EIS) afin de mettre à jour les informations de contact SAP en conséquence.

Comme les structures de données utilisées pour le stockage de contacts sont différentes dans les systèmes Siebel et SAP, le composant SCA ContactSync doit assurer le mappage.

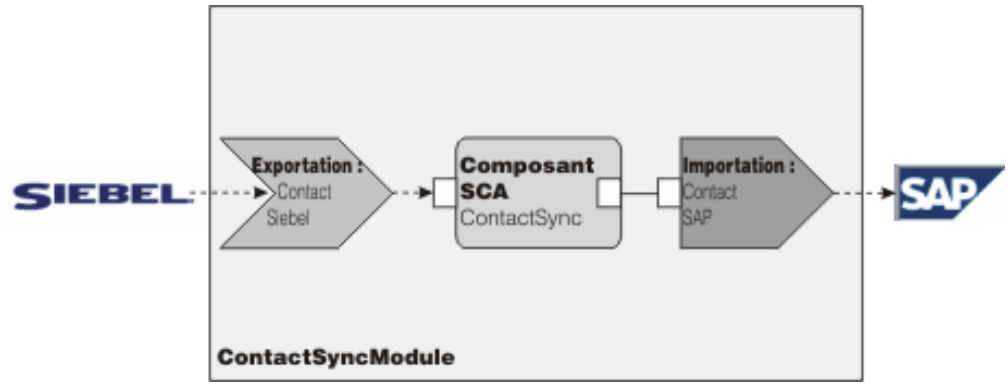


Figure 32. Flux provenant d'un système Siebel vers un système SAP

L'exportation Siebel Contact et l'importation SAP Contact ont les adaptateurs de ressources appropriés configurés.

Principales fonctionnalités des liaisons EIS

Une importation EIS est une importation SCA (Service Component Architecture) qui permet aux composants du module SCA d'utiliser les applications EIS définies hors du module SCA. Une importation EIS permet de transférer des données d'un composant SCA vers un système EIS externe. Une exportation EIS permet de transférer des données d'un système EIS externe vers un module SCA.

Importations

La fonction de l'importation EIS est de combler l'écart entre les composants SCA et les systèmes EIS externes. Les applications externes peuvent être traitées comme des importations EIS. Dans ce cas, l'importation EIS envoie des données au système EIS externe et reçoit éventuellement des données en réponse.

L'importation EIS fournit aux composants SCA une vue uniforme des applications externes au module. Ceci permet aux composants de communiquer avec un système EIS externe, tel que SAP, Siebel ou PeopleSoft, via un modèle de programmation SCA homogène.

Côté client de l'importation, une interface est affichée par l'application d'importation EIS, laquelle comporte une ou plusieurs méthodes, chacune prenant des objets de données servant d'arguments et de valeurs de retour. Côté implémentation, une interface client commune (Common Client Interface - CCI) est implémentée par un adaptateur de ressources.

L'implémentation d'exécution d'une importation EIS connecte l'interface côté client et l'interface CCI. L'importation mappe l'appel de la méthode de l'interface à l'appel sur l'interface CCI.

Des liaisons sont créées à trois niveaux : une liaison d'interface, qui utilise les liaisons de méthode incluses, lesquelles utilisent les liaisons de données.

La liaison d'interface associe l'interface de l'importation à la connexion au système EIS qui fournit l'application. Ceci reflète le fait que l'ensemble d'applications, représenté par l'interface, est fourni par l'instance spécifique du système EIS et que la connexion permet l'accès à cette instance. L'élément de liaison contient des propriétés avec suffisamment d'informations pour créer la connexion (ces propriétés font partie de l'instance javax.resource.spi.ManagedConnectionFactory).

La liaison de méthode associe la méthode à l'interaction spécifique avec le système EIS. Pour JCA, cette interaction se caractérise par l'ensemble de propriétés de l'implémentation de l'interface javax.resource.cci.InteractionSpec. L'élément d'interaction de la liaison de méthode contient ces propriétés, ainsi que le nom de la classe, fournissant ainsi assez d'informations pour permettre l'interaction. La liaison de méthode utilise des liaisons de données décrivant le mappage de l'argument et du résultat de la méthode d'interface à la représentation EIS.

Le scénario d'exécution d'une importation EIS est le suivant :

1. La méthode de l'interface d'importation est appelée à l'aide du modèle de programmation SCA.
2. La demande, qui parvient au gestionnaire d'importation EIS, contient le nom de la méthode et ses arguments.
3. Tout d'abord, l'importation crée une implémentation de liaison d'interface. Ensuite, à partir des données de la liaisond'importation, il crée une ConnectionFactory et associe les deux. Autrement dit, l'importation appellesetConnectionFactory sur la liaison d'interface.
4. L'implémentation de liaison de méthode correspondant à la méthode appelée est créée.
5. L'instance javax.resource.cci.InteractionSpec est créée et remplie, puis les liaisons de données sont utilisées pour lier les arguments de la méthode à un format lisible par l'adaptateur de ressource.
6. L'interface CCI permet d'appliquer l'interaction.
7. Lors du retour de l'appel, la liaison de données permet de créer le résultat de l'appel et de renvoyer ce résultat au demandeur.

Exportations

La fonction de l'exportation EIS est de combler l'écart entre un composant SCA et un système EIS externe. Les applications externes peuvent être traitées comme des exportations EIS. Dans ce cas, l'application externe envoie ses données sous la forme de notifications périodiques. Une exportation EIS peut être considérée comme une application d'abonnement qui écoute une demande externe provenant d'un EIS. Le composant SCA qui utilise l'exportation EIS la voit comme une application locale.

L'exportation EIS fournit aux composants SCA une vue uniforme des applications externes au module. Ceci permet aux composants de communiquer avec un système EIS, tel que SAP, Siebel ou PeopleSoft, via un modèle de programmation SCA homogène.

L'exportation comporte une implémentation d'écouteur recevant des demandes du système EIS. L'écouteur implémente une interface d'écouteur spécifique à l'adaptateur de ressources. L'exportation contient également une interface d'implémentation de composant, exposée au système EIS via l'exportation.

L'implémentation d'exécution d'une exportation EIS connecte l'écouteur et l'interface d'implémentation de composant. L'exportation mappe la demande EIS à l'appel de l'opération appropriée sur le composant. Des liaisons sont créées à trois niveaux : une liaison d'écouteur, qui utilise une liaison de méthode native incluse, laquelle utilise une liaison de données.

La liaison d'écouteur associe l'écouteur qui reçoit des demandes au composant affiché via l'exportation. La définition d'exportation contient le nom du composant ; l'environnement d'exécution le localise et lui transmet des demandes.

La liaison de méthode native associe la méthode native ou le type d'événement reçu par l'écouteur à l'opération implémentée par le composant affiché au moyen de l'exportation. Il n'y a aucune relation entre la méthode appelée sur l'écouteur et le type d'événement ; tous les événements arrivent par l'intermédiaire d'une ou de plusieurs méthodes de l'écouteur. La liaison de méthode native utilise le sélecteur de fonction défini dans l'exportation pour extraire le nom de la méthode native des données entrantes et des liaisons de données pour lier le format de données du système EIS à un format lisible par le composant.

Le scénario d'exécution d'une exportation EIS est le suivant :

1. La demande EIS déclenche l'appel de la méthode sur l'implémentation de l'écouteur.
2. L'écouteur localise et appelle l'application d'exportation en lui transmettant tous les arguments d'appel.
3. L'exportation crée l'implémentation de liaison d'écouteur.
4. L'exportation instancie le sélecteur de fonction et le définit sur la liaison d'écouteur.
5. L'exportation initialise les liaisons de méthode native et les ajoute à la liaison de l'écouteur. Pour chaque liaison de méthode native, les liaisons de données sont également initialisées.
6. L'exportation appelle la liaison de l'écouteur.
7. Cette liaison localise les composants exportés et utilise le sélecteur de fonction pour extraire le nom de la méthode native.
8. Ce nom est utilisé pour localiser la liaison de méthode native, qui peut alors appeler le composant cible.

Le type d'interaction de l'adaptateur permet à la liaison d'exportation EIS d'appeler le composant cible de manière asynchrone (par défaut) ou synchrone.

Adaptateurs de ressources

Vous développez une importation ou une exportation à l'aide de l'assistant de service externe puis, lors du développement, vous incluez un adaptateur de ressources. Les adaptateurs livrés avec WebSphere Integration Developer pour accéder aux systèmes CICS, IMS, JD Edwards, PeopleSoft, SAP et Siebel sont réservés à des fins de développement et de test uniquement. Autrement dit, vous les utilisez pour développer et tester vos applications.

Après le déploiement de votre application, vous avez besoin d'adaptateurs d'exécution sous licence pour exécuter votre application. Toutefois, lorsque vous créez votre service, vous pouvez y intégrer l'adaptateur. La licence pour votre adaptateur peut vous permettre d'utiliser l'adaptateur intégré comme adaptateur d'exécution sous licence. Ces adaptateurs sont compatibles avec la norme JCA 1.5 (Java EE Connector Architecture). JCA, standard ouvert, est la norme Java EE pour la connectivité EIS. JCA fournit un cadre géré ; la qualité de service (QoS) est assurée par le serveur d'applications qui offre aux transactions la sécurité et la gestion du cycle de vie. Ces adaptateurs sont également compatibles avec la spécification Enterprise Metadata Discovery, à l'exception de l'adaptateur de ressources IBM CICS ECI et du connecteur IBM IMS pour Java.

Les adaptateurs WebSphere Business Integration Adapters, un ancien jeu d'adaptateurs, sont également pris en charge par l'assistant.

Ressources Java EE

Le module EIS, un module SCA qui repose sur le modèle des modules EIS, peut être déployé sur la plateforme Java EE.

Le déploiement d'un module EIS sur la plateforme Java EE génère une application prête à être exécutée, encapsulée dans un fichier EAR et déployée sur le serveur. Tous les artefacts et ressources Java EE sont créés ; l'application est configurée et prête pour l'exécution.

Propriétés dynamiques des spécifications d'interaction et de connexion JCA

La liaison EIS peut accepter des données en entrée pour lesspécifications InteractionSpec et ConnectionSpec spécifiées, en utilisant un objet de données enfant bien défini qui accompagne la charge. Ceci permet des interactions demande-réponse dynamiques avec un adaptateur de ressources par le biais de InteractionSpec et l'authentification des composants par le biais de ConnectionSpec.

L'interface javax.cci.InteractionSpec transmet des informations sur le mode de traitement de la demande d'interaction avec l'adaptateur de ressources. Elle comporte également des informations sur l'accomplissement de l'interaction après la demande. Ces communications bidirectionnelles par le biais des interactions sont parfois appelées *conversations*.

La liaison EIS s'attend à ce que la charge qui sera l'argument de l'adaptateur de ressources contienne un objet de données enfant appelé `properties`. Cette objet de données de propriétés contient des paires nom/valeur, avec le nom des propriétés de la spécification d'interaction dans un format particulier. Les règles de formatage sont les suivantes :

- Les noms doivent commencer par le préfixe `IS`, suivi du nom de propriété. Par exemple, une spécification d'interaction avec une propriété JavaBeans appelée `InteractionId` doit spécifier le nom de la propriété sous la forme `ISInteractionId`.
- La paire nom/valeur représente le nom et la valeur du type simple de propriété de la spécification d'interaction.

Dans cet exemple, une interface spécifie que l'entrée d'une opération est un objet de données `Compte`. Cette interface appelle une application de liaison d'importation EIS dans le but d'envoyer et de recevoir une propriété InteractionSpec dynamique appelée `workingSet` avec la valeur `xyz`.

Le graphique métier ou les objets métier du serveur contiennent un objet métier `properties` sous-jacent qui permet l'envoi de données propres au protocole avec la charge. Cet objet métier`properties` est intégré, ce qui fait qu'il n'est pas nécessaire de le spécifier dans le schéma XML lors de la construction d'un objet métier. Il convient seulement de le créer et de l'utiliser. Si vous avez défini vos propres types de données sur la base du schéma XML, vous devez spécifier un élément `properties` qui contient les paires nom/valeur attendues.

```
BOFactory dataFactory = (BOFactory) \
serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
 //Wrapper for doc-lit wrapped style interfaces,
```

```
  //skip to payload for non doc-lit
  DataObject docLitWrapper = dataFactory.createByElement /
  ("http://mytest/eis/Account", "AccountWrapper");
```

Créez la charge.

```
DataObject account = docLitWrapper.createDataObject(0);
  DataObject accountInfo = account.createDataObject("AccountInfo");
  //Perform your setting up of payload


  //Construct properties data for dynamic interaction

  DataObject properties = account.createDataObject("properties");
```

Pour le nom workingSet, définissez la valeur attendue (xyz).

```
properties.setString("ISworkingSet", "xyz");


  //Invoke the service with argument

  Service accountImport = (Service) \
  serviceManager.locateService("AccountOutbound");
  DataObject result = accountImport.invoke("createAccount", docLitWrapper);

  //Get returned property
  DataObject retProperties = result.getDataObject("properties");

  String workingset = retProperties.getString("ISworkingSet");
```

Vous pouvez utiliser des propriétés ConnectionSpec pour l'authentification des composants dynamiques. Les mêmes règles s'appliquent que ci-dessus, sauf que le préfixe du nom de propriété doit être `CS` au lieu de `IS`. Les propriétés ConnectionSpec ne sont pas bidirectionnelles. Le même objet de données `properties` peut contenir à la fois des propriétés IS et CS.

Pour utiliser les propriétés ConnectionSpec, définissez `resAuth` spécifié dans la liaison d'importation sur `Application`. Assurez-vous également que l'adaptateur de ressources prend en charge l'autorisation de composant. Pour plus de détails, reportez-vous au chapitre 8 du document J2EE Connector Architecture Specification.

Clients externes et liaisons EIS

Le serveur peut envoyer des messages à, ou recevoir des messages de clients externes par le biais de liaisons EIS.

Un client externe, par exemple un portail Web ou un système EIS doit envoyer un message à un module SCA dans le serveur ou doit être appelé par un composant à partir du serveur.

Le client appelle l'importation EIS comme avec toute autre application, à l'aide soit de l'interface DII (Dynamic Invocation Interface), soit de l'interface Java.

1. Le client externe crée une instance de l'interface ServiceManager et recherche l'importation EIS à l'aide de son nom de référence. Le résultat de la recherche est une implémentation de l'interface de service.
2. Le client crée un argument d'entrée, un objet de données générique, créé dynamiquement à l'aide du schéma de l'objet de données. Cette étaoe est effectuée à l'aide de l'implémentation de l'interface Service Data Object DataFactory.
3. Le client externe appelle l'EIS et obtient les résultats requis.

Le client a également la possibilité d'appeler l'importation EIS par le biais de l'interface Java.

1. Le client externe crée une instance de l'interface ServiceManager et recherche l'importation EIS à l'aide de son nom de référence. Le résultat de la recherche est une interface Java de l'importation EIS.
2. Le client crée un argument d'entrée et un objet de données typé.
3. Le client appelle l'EIS et obtient les résultats requis.

L'interface d'exportation EIS définit l'interface du composant SCA exporté qui est accessible aux applications EIS externes. Cette interface peut être considérée comme l'interface qu'une application externe (comme SAP ou PeopleSoft) va appeler par le biais de l'implémentation de l'environnement d'exécution de l'application d'exportation EIS.

L'exportation utilise EISExportBinding pour lier les services exportés à l'application EIS externe. Elle vous permet d'abonner une application contenue dans votre module SCA pour écouter les demandes de service EIS. La liaison d'exportation EIS spécifie le mappage entre la définition des événements entrants telle qu'elle est comprise par l'adaptateur de ressources (à l'aide des interfaces Java EE Connector Architecture) et l'appel des opérations SCA.

L'interface EISExportBinding exige que les services EIS externes soient basés sur des contrats entrants de Java EE Connector Architecture 1.5. L'interface EISExportBinding exige qu'un gestionnaire de données ou une liaison de données soit spécifié(e) au niveau de la liaison ou au niveau de la méthode.

Liaisons JMS

Un fournisseur JMS (Java Message Service) active la messagerie en fonction du modèle de programmation et de l'API JMS (Java Messaging Service). Il fournit des fabriques de connexions Java EE pour créer des connexions pour des destinations JMS et pour envoyer et recevoir des messages.

Trois liaisons JMS sont fournies :

- Liaison fournisseur SIB (bus d'intégration de services) conforme JMS JCA 1.5 (*liaisons JMS*)
- Liaison JMS non JCA générique conforme à JMS 1.1 (*Liaison JMS générique*)
- Liaison JMS WebSphere MQ, fournissant une prise en charge fournisseur JMS pour WebSphere MQ et permettant l'interopérabilité avec les applications Java EE (liaison JMS *WebSphere MQ*)

Les liaisons d'importation et d'exportation JMS permettent à un module SCA (Service Component Architecture) d'effectuer des appels et de recevoir des messages à partir de systèmes JMS externes.

Les liaisons WebSphere MQ (liaison *WebSphere MQ*) qui permettent aux utilisateurs MQ natifs de traiter des formats de message entrant et sortant arbitraires (WebSphere MQ requis).

Les liaisons d'importation et d'exportation JMS permettent une intégration aux applications JMS en utilisant le fournisseur JMS SIB basé sur JCA 1.5 qui fait partie de WebSphere Application Server. D'autres adaptateurs de ressources JMS basés sur JCA 1.5 ne sont pas pris en charge

Concepts associés

«Présentation des liaisons JMS»
Les liaisons JMS assurent la connectivité entre l'environnement SCA (Service Component Architecture) et les systèmes JMS.

«Intégration JMS et adaptateurs de ressources», à la page 89
Le service JMS (Java Message Service) assure une intégration par le biais d'un adaptateur de ressources JMS disponible basé sur JCA 1.5. La prise en charge complète de l'intégration JMS est assurée pour l'adaptateur de ressources JMS du bus d'intégration de services (SIB).

«En-têtes JMS», à la page 90
Un message JMS contient deux types d'en-têtes – l'en-tête de système JMS et plusieurs propriétés JMS. Il est possible d'accéder aux deux types d'en-tête depuis un module de médiation dans l'objet SMO (Service Message Object) ou en utilisant l'API ContextService.

«Schéma de corrélation des destinations de réponse dynamique temporaires JMS», à la page 92
Le schéma de corrélation des destinations de réponse dynamique temporaires crée une rubrique ou une file d'attente dynamique unique pour chaque demande envoyée.

«Clients externes», à la page 92
Le serveur peut envoyer des messages à, ou recevoir des messages de clients externes par le biais de liaisons JMS.

«Identification et résolution des incidents liés aux liaisons JMS», à la page 94
Vous pouvez diagnostiquer et résoudre des incidents survenant sur les liaisons JMS.

«Gestion des exceptions», à la page 95
La configuration de la liaison détermine la manière dont sont gérées les exceptions qui sont émises par les gestionnaires de données ou les liaisons de données. En outre, la nature du flux de médiation détermine le comportement du système lorsqu'une exception de ce type est générée.

Référence associée

«Principales fonctionnalités des liaisons JMS», à la page 89
Les fonctionnalités essentielles de l'importation et l'exportation JMS incluent les en-têtes et les ressources Java EE créées.

Présentation des liaisons JMS

Les liaisons JMS assurent la connectivité entre l'environnement SCA (Service Component Architecture) et les systèmes JMS.

Liaisons JMS

Les principaux composants des liaisons d'importation et d'exportation JMS sont les suivants :

- Adaptateur de ressources : assure une connectivité bidirectionnelle gérée entre un module SCA et des systèmes JMS externes
- Connexions : encapsulent une connexion virtuelle entre un client et une application fournisseur
- Destinations : utilisées par un client pour spécifier la cible des messages produits ou la source des messages utilisés
- Données d'authentification : permettent de sécuriser l'accès à la liaison

Liaisons d'importation JMS

Les liaisons d'importation JMS vous permettent d'importer une application JMS externe que vous pourrez utiliser dans votre module SCA. Les liaisons d'importation JMS permettent aux composants au sein de votre module SCA de communiquer avec les services fournis par les applications JMS externes.

Les connexions avec le fournisseur JMS associé de destinations JMS sont créées à l'aide d'une fabrique de connexions JMS. Utilisez les objets d'administration de fabrique de connexions afin de gérer des fabriques de connexion JMS pour le fournisseur de messagerie par défaut.

L'interaction avec les systèmes JMS externes comprend l'utilisation des destinations pour l'envoi des requêtes et la réception des réponses.

Deux types de scénarios d'utilisation pour l'importation JMS sont pris en charge en fonction du type d'opération appelée :

- Unidirectionnel : l'importation JMS place un message sur la destination d'envoi configurée dans la liaison d'importation. Rien n'est défini dans la zone replyTo de l'en-tête JMS.
- Bidirectionnel (demande-réponse) : l'importation JMS place un message sur la destination d'envoi et conserve ensuite la réponse reçue depuis le composant SCA.

  La liaison d'importation peut être configurée (à l'aide de la zone **Schéma de corrélation de réponse** dans WebSphere Integration Developer) pour faire en sorte que l'ID de corrélation de message de réponse soit copié à partir de l'ID de message de demande (valeur par défaut), ou à partir de l'ID de corrélation de message de demande. La liaison d'importation peut également être configurée pour utiliser une destination de réponse dynamique temporaire afin d'établir une corrélation entre les réponses et les demandes. Une destination temporaire est créée pour chaque demande et l'importation l'utilise pour recevoir la réponse.

  La destination receive est définie dans la propriété d'en-tête replyTo du message sortant. Un écouteur de messages est déployé pour écouter sur la destination de réception et, dès qu'une réponse est reçue, l'écouteur de messages transmet la réponse au composant.

Pour les scénarios d'utilisation unidirectionnel et bidirectionnel, les propriétés d'en-tête dynamique et statique peuvent être spécifiées. Les propriétés statiques peuvent être définies à partir de la liaison de méthode d'importation JMS. Certaines de ces propriétés revêtent des significations particulières pour l'environnement d'exécution JMS SCA.

Il est important de noter que JMS est une liaison asynchrone. Si un composant appelant appelle une importation JMS de manière synchrone (pour une opération bidirectionnelle), le composant appelant est bloqué jusqu'à ce que la réponse soit renvoyée par le service JMS.

La figure 33, à la page 88 illustre la manière dont l'importation est liée au service externe.

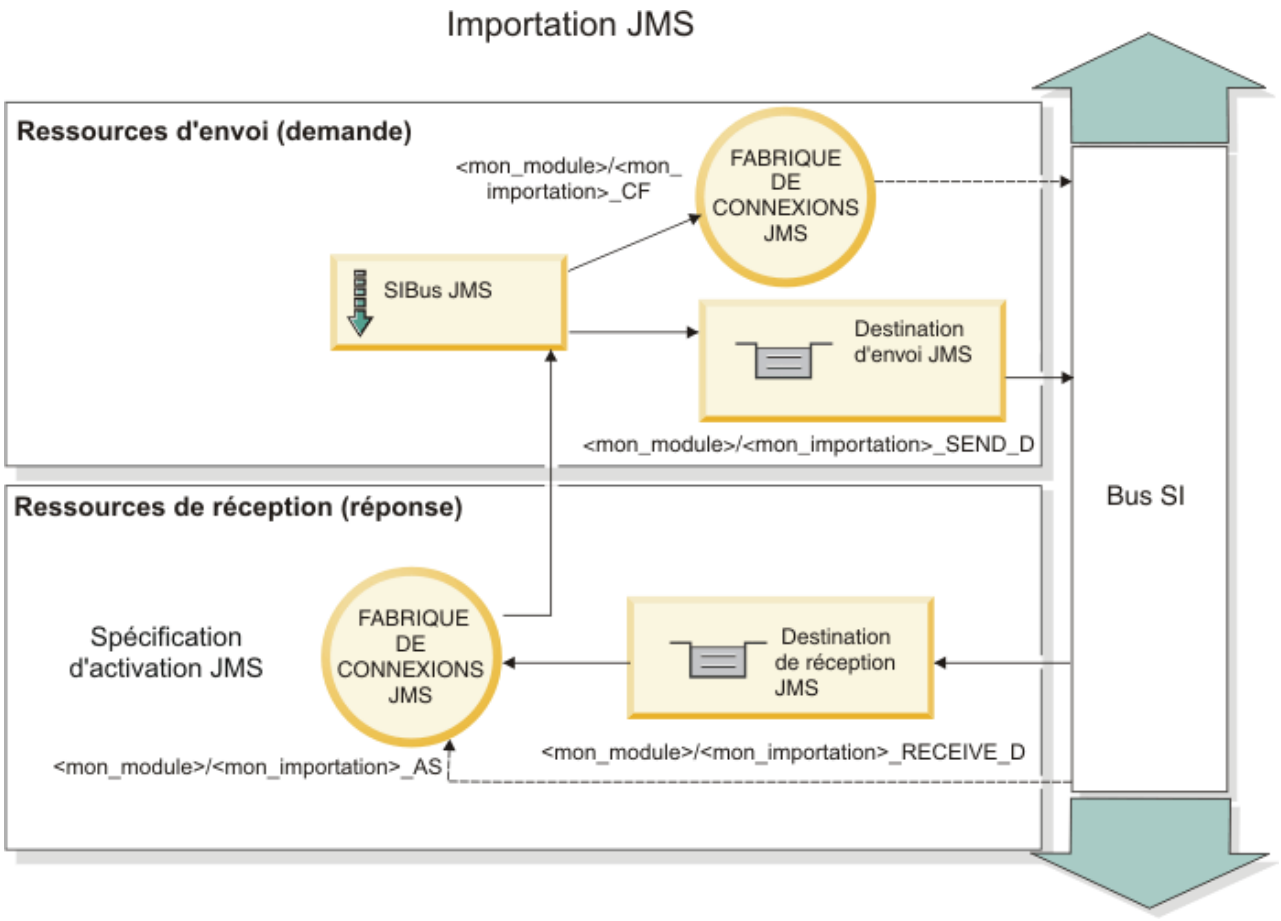


Figure 33. Ressources de liaisons d'importation JMS

Liaisons d'exportation JMS

Les liaisons d'exportation JMS offrent les moyens aux modules SCA de fournir des services aux applications JMS externes.

La connexion qui fait partie d'une exportation JMS est une spécification d'activation.

Une exportation JMS comporte des destinations d'envoi et de réception.

- La destination receive est le lieu de réception du message entrant destiné au composant cible.
- La destination send est celle à laquelle la réponse sera envoyée, saus si le message entrant l'a remplacé en utilisant la propriété d'en-tête replyTo.

Un écouteur de messages est déployé pour écouter les demandes parvenant à la destination receive spécifiée dans la liaison d'exportation. La destination spécifiée dans la zone send est utilisée pour envoyer la réponse à la demande entrante si le composant appelé fournit une réponse. La destination spécifiée dans la zone replyTo du message entrant remplace la destination spécifiée dans send.

La figure 34, à la page 89 illustre la manière dont le demandeur externe est lié à l'exportation.

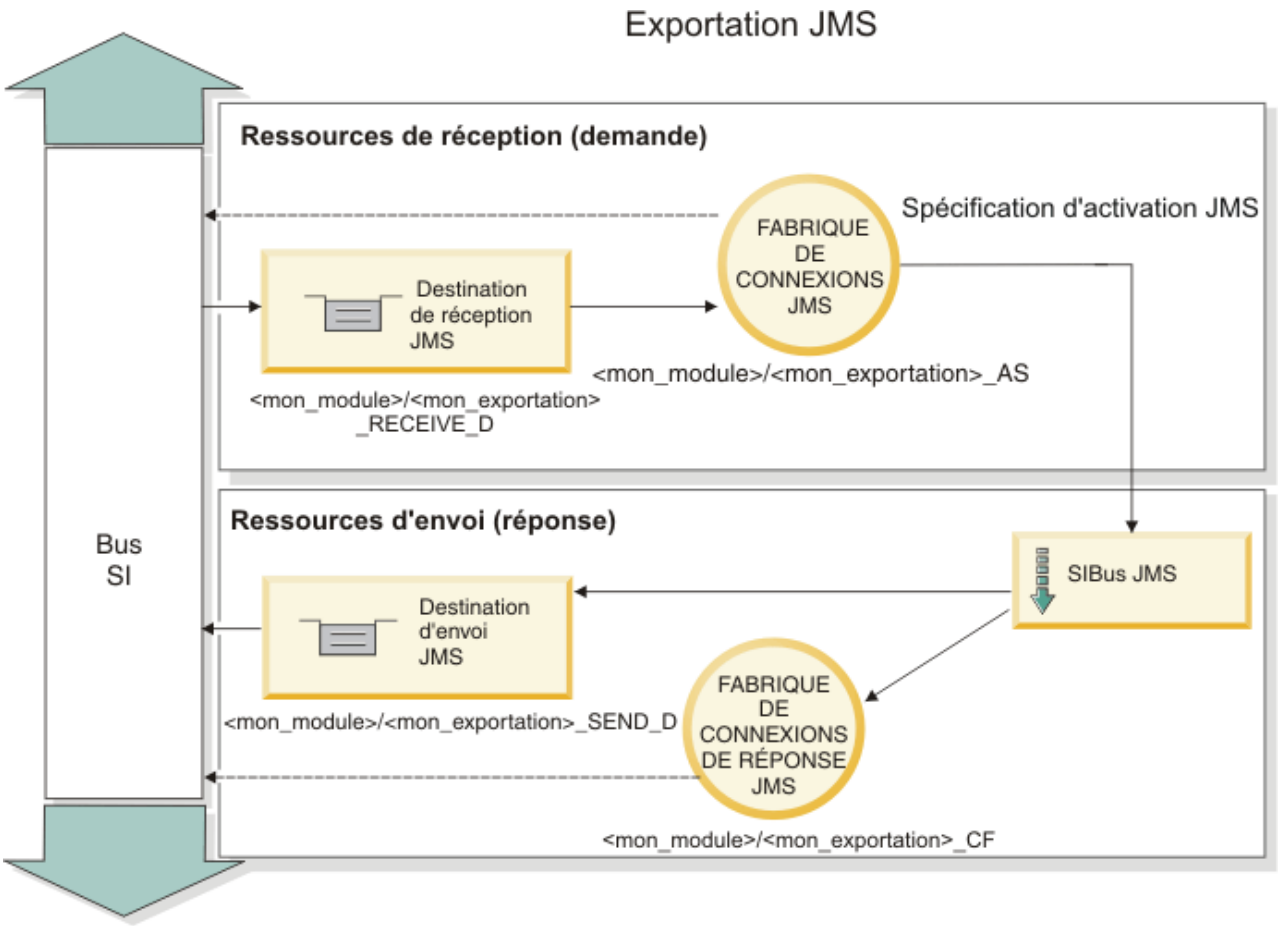


Figure 34. Ressources de liaisons d'exportation JMS

Intégration JMS et adaptateurs de ressources

Le service JMS (Java Message Service) assure une intégration par le biais d'un adaptateur de ressources JMS disponible basé sur JCA 1.5. La prise en charge complète de l'intégration JMS est assurée pour l'adaptateur de ressources JMS du bus d'intégration de services (SIB).

Utilisez un fournisseur JMS pour JCA 1.5 lorsque vous souhaitez une intégration avec un système JMS externe conforme au JCA 1.5. Les services externes conformes au JCA 1.5 peuvent recevoir et envoyer des messages à intégrer avec vos composants SCA à l'aide de l'adaptateur de ressources JMS SIB.

L'emploi d'adaptateurs de ressources JCA 1.5 propres à un fournisseur n'est pas pris en charge.

Il est impossible de déployer des modules JMS dans un environnement Java SE. En effet, ils ne peuvent être déployés que dans un environnement Java EE.

Principales fonctionnalités des liaisons JMS

Les fonctionnalités essentielles de l'importation et l'exportation JMS incluent les en-têtes et les ressources Java EE créées.

En-têtes spéciaux

Des propriétés d'en-tête spéciales sont utilisées dans les importations et les exportations JMS pour indiquer à la cible comment traiter le message.

Par exemple, TargetFunctionName mappe la méthode native à la méthode d'opération.

Ressources Java EE

Plusieurs ressources Java EE sont créées lorsque les importations et les exportations JMS sont déployées dans un environnement Java EE.

ConnectionFactory
Utilisée par les clients pour créer une connexion au fournisseur JMS.

ActivationSpec
Utilisé par les importations pour la réception de la réponse à une demande et par les exportations pour la configuration endpoint des noeuds finaux de message qui représentent les écouteurs de messages dans leurs interactions avec le système de messagerie.

Destinations

- Destination d'envoi : Destination à laquelle est envoyé le message de demande ou sortant (importation) ou le message de réponse (exportation), si la valeur n'est pas remplacée par la zone d'en-tête JMSReplyTo du message entrant.
- Destination de réception : emplacement où doit être placé le message entrant ; dans les importations, il s'agit d'une réponse, dans les exportations, il s'agit d'une demande.
- Destination de rappel : Destination du système JMS SCA utilisée pour stocker les informations de corrélation. Vous ne devez pas procéder à des opérations de lecture ou d'écriture sur cette destination.

La tâche d'installation crée ConnectionFactory et trois destinations. En outre, elle crée ActivationSpec pour permettre à l'écouteur de messages d'exécution d'écouter les réponses sur la destination de réception. Les propriétés de ces ressources sont définies dans le fichier d'importation ou d'exportation.

En-têtes JMS

Un message JMS contient deux types d'en-têtes – l'en-tête de système JMS et plusieurs propriétés JMS. Il est possible d'accéder aux deux types d'en-tête depuis un module de médiation dans l'objet SMO (Service Message Object) ou en utilisant l'API ContextService.

En-tête système JMS

L'en-tête système JMS est représenté dans l'objet SMO par l'élément JMSHeader qui contient toutes les zones généralement présentes dans un en-tête JMS. Bien qu'elles puissent être modifiées dans la médiation (ou ContextService), certaines zones d'en-tête système JMS définies dans l'objet SMO ne seront pas propagées dans le message JMS sortant puisqu'elles sont remplacées par des valeurs statiques ou système.

Les zones clés de l'en-tête système JMS qui peuvent être mises à jour dans une médiation (ou ContextService) sont :

- **JMSType** et **JMSCorrelationID** : valeurs des propriétés de l'en-tête du message prédéfinies spécifiques
- **JMSDeliveryMode** – valeurs du mode de livraison (persistant (par défaut) ou non persistant)
- **JMSPriority** : valeur de la priorité (0 à 9 ; la valeur par défaut est JMS_Default_Priority)

Propriétés JMS

Les propriétés JMS sont représentées dans l'objet SMO en tant qu'entrées dans la liste Propriétés. Les propriétés peuvent être ajoutées, mises à jour ou supprimées dans une médiation ou en utilisant l'API ContextService.

Elles peuvent également être définies de manière statique dans la liaison JMS. Les propriétés définies de manière statique remplacent les paramètres (portant le même nom) qui sont définis de manière dynamique.

Les propriétés utilisateur propagées à partir d'autres liaisons (par exemple, une liaison HTTP) correspondront à une sortie dans la liaison JMS comme les propriétés JMS.

Paramètres de propagation d'en-tête

La propagation des propriétés et des en-têtes système JMS depuis le message JMS entran t vers les composants en aval ou depuis les composants en amont vers le message JMS sortant peut être contrôlée par l'indicateur Propagate Protocol Header.

Lorsque Propagate Protocol Header est défini, les informations d'en-tête peuvent circuler vers le message ou vers le composant cible, comme indiqué dans la liste suivante :

- Requête d'exportation JMS

  L'en-tête JMS reçu dans le message sera propagé aux composants cible via le service de contexte. Les propriétés JMS reçues dans le message seront propagées aux composants cible via le service de contexte.
- Réponse d'exportation JMS

  Toute zone d'en-tête JMS définie dans le service de contexte sera utilisée dans le message sortant, excepté en cas de remplacement par les propriétés statiques définies sur la liaison d'exportation JMS. Toute propriété définie dans le service de contexte sera utilisée dans le message sortant excepté en cas de remplacement par les propriétés statiques définies sur la liaison d'exportation JMS.
- Requête d'importation JMS

  Toute zone d'en-tête JMS définie dans le service de contexte sera utilisée dans le message sortant, excepté en cas de remplacement par les propriétés statiques définies sur la liaison d'importation JMS. Toute propriété définie dans le service de contexte sera utilisée dans le message sortant excepté en cas de remplacement par les propriétés statiques définies sur la liaison d'importation JMS.
- Réponse d'importation JMS

  L'en-tête JMS reçu dans le message sera propagé aux composants cible via le service de contexte. Les propriétés JMS reçues dans le message seront propagées aux composants cible via le service de contexte.

Schéma de corrélation des destinations de réponse dynamique temporaires JMS

Le schéma de corrélation des destinations de réponse dynamique temporaires crée une rubrique ou une file d'attente dynamique unique pour chaque demande envoyée.

La destination de réponse statique spécifiée dans l'importation permet de déterminer la nature de la rubrique ou de la file d'attente de destination temporaire. Elle est définie dans la zone **ReplyTo** de la demande et l'importation JMS écoute les réponses sur cette destination. Lors de la réception d'une réponse, cette dernière est replacée dans la file d'attente de la destination des réponses statiques pour un traitement asynchrone. La zone **CorrelationID** de la réponse n'est pas utilisée et n'a pas besoin d'être définie.

Incidents transactionnels

Si une destination dynamique temporaire est utilisée, la réponse doit être consommée dans la même unité d'exécution que la réponse envoyée. La demande doit être envoyée en dehors de la transaction globale et doit être validée avant d'être reçue par le service dorsal et avant qu'une réponse ne soit renvoyée.

Persistance

Les files d'attente dynamiques temporaires sont des entités dont la durée de vie est courte et ne garantissent pas le même niveau de persistance que celui associé à une rubrique ou une file d'attente statique. Une rubrique ou une file d'attente dynamique temporaire ne survit pas à un redémarrage du serveur, tout comme les messages. Une fois que le message a été replacé en file d'attente dans la destination de réponse statique, elle conserve la persistance définie dans le message.

Délai d'attente

L'importation attend de recevoir la réponse sur la destination de réponse dynamique temporaire pendant un certain temps. Ce délai est extrait du qualifiant de délai d'expiration des réponses SCA, s'il est défini. Dans le cas contraire, le délai par défaut est de 60 secondes. En cas de dépassement de ce délai, l'importation génère une erreur `ServiceTimeoutRuntimeException`.

Clients externes

Le serveur peut envoyer des messages à, ou recevoir des messages de clients externes par le biais de liaisons JMS.

Un client externe (comme un portail Web ou un système d'information d'entreprise) peut envoyer un message à un module SCA dans le serveur ou peut être appelé par un composant à partir du serveur.

Les composants d'exportation JMS déploient des écouteurs des messages pour écouter les demandes entrantes sur la destination receive spécifié dans la liaison d'exportation. La destination spécifiée dans la zone send est utilisée pour envoyer la réponse à la demande entrante si l'application appelée fournit une réponse. Ainsi, un client externe est en mesure d'appeler des applications avec la liaison d'exportation.

Les importations JMS effectuent des liaisons avec des clients externes et peuvent ainsi leur envoyer des messages. Ces messages peuvent ou non exiger une réponse de la part du client externe.

Tâches associées

«Utilisation de clients externes»
Un client externe (c'est-à-dire extérieur au serveur) peut avoir besoin d'interagir avec une application installée sur le serveur.

Utilisation de clients externes :

Un client externe (c'est-à-dire extérieur au serveur) peut avoir besoin d'interagir avec une application installée sur le serveur.

task_context

Le scénario présenté ici est très simple : un client externe souhaite interagir avec une application sur le serveur. La figure représente un scénario simple typique.

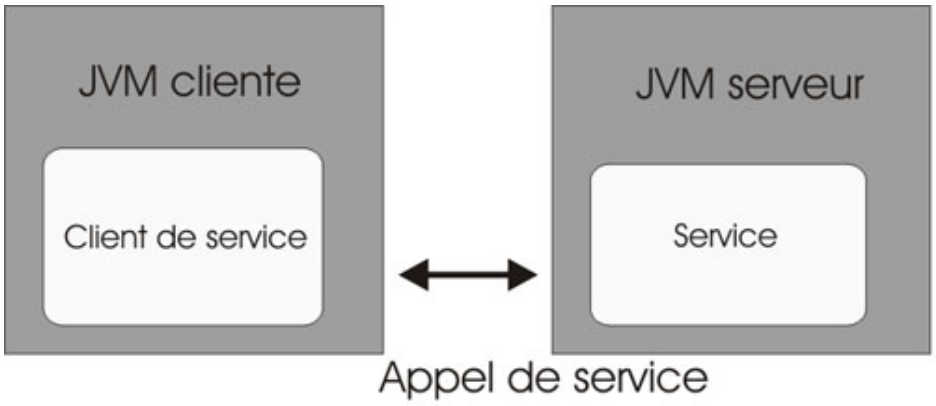


Figure 35. Scénario simple : interaction entre un client externe et une application du serveur

L'application SCA inclut une exportation avec une liaison JMS ; c'est ce qui la rend disponible aux clients externes.

Si un client externe se trouve sur une machine Java virtuelle (JVM) distincte de votre serveur, vous devez effectuer plusieurs opérations afin d'établir la connexion et permettre l'interaction avec une exportation JMS. Le client obtient un contexte initial (InitialContext) avec les valeurs appropriées, puis il compulse les ressources via JNDI. Le client utilise alors le client de spécification JMS 1.1 pour accéder aux destinations et envoyer/recevoir des messages sur les destinations.

Les noms JNDI par défaut des ressources créées automatiquement par l'environnement d'exécution sont répertoriés dans la rubrique configuration de cette section. Toutefois, si vous avez pré-créé les ressources, utilisez ces noms JNDI.

task_procedure

1. Configurer les destinations JMS et la fabrique de connexions pour envoyer le message.
2. Vérifier que le contexte JNDI, le port de l'adaptateur de ressources SIB et le port d'amorçage de la messagerie sont corrects.

   Le serveur utilise certains ports par défaut, mais si plusieurs serveurs sont installés sur ce système, des ports de remplacement sont créés lors de l'installation pour éviter les conflits avec d'autres instances de serveur. Vous pouvez utiliser la console d'administration pour déterminer quels ports votre

serveur employer. Accédez à **Serveurs** → **Serveurs d'applications** → *nom_serveur* → **Configuration** et cliquez sur **Ports** sous **Communication**. Vous pouvez maintenant modifier le port utilisé.

3. Le client obtient un contexte initial avec les valeurs appropriées, puis il compulse les ressources via JNDI.
4. A l'aide des spécifications JMS 1.1, le client accède aux destinations et aux messages envoyés et reçus sur les destinations.

Identification et résolution des incidents liés aux liaisons JMS

Vous pouvez diagnostiquer et résoudre des incidents survenant sur les liaisons JMS.

Exceptions liées à l'implémentation

En réponse à diverses conditions d'erreur, l'implémentation de l'importation et de l'exportation JMS peut renvoyer deux types d'exception :

- Service : cette exception est renvoyée si l'erreur définie sur l'interface métier de service (type de port WSDL) s'est produite.
- Service Runtime : générée dans tous les autres cas. Dans la plupart des cas, l'exception cause contient l'exception d'origine (JMSException).

  Par exemple, une importation n'attend qu'un seul message de réponse pour chaque message de demande. Si plusieurs réponses sont fournies, ou si une réponse tardive (une pour laquelle la réponse SCA a expiré) est fournie, une exception Service Runtime est générée. La transaction est annulée et le message de réponse est retiré de la file d'attente ou traité par le gestionnaire d'événements ayant échoué.

Conditions d'erreur principales

Les principales conditions d'erreur liées aux liaisons JMS sont déterminées par la sémantique transactionnelle, la configuration du fournisseur JMS ou une référence au fonctionnement existant dans d'autres composants. Les causes premières d'incident peuvent être :

- Erreur de connexion au fournisseur ou à la destination JMS.

  Une erreur de connexion au fournisseur JMS pour recevoir des messages entraîne l'échec de démarrage de l'écouteur des messages. Cette situation sera enregistrée dans la journal WebSphere Application Server. Des messages persistants resteront sur la destination jusqu'à ce qu'ils soient récupérés (ou qu'ils parviennent à échéance).

  Une erreur de connexion au fournisseur JMS destinée à envoyer des messages entraîne l'annulation de la transaction qui contrôle l'envoi.
- Erreur d'analyse d'un message entrant ou de construction d'un message sortant.

  Une erreur de liaison des données ou du gestionnaire de données entraîne l'annulation de la transaction qui contrôle le travail.
- Erreur d'envoi du message sortant.

  Un échec d'envoi d'un message entraîne l'annulation de la transaction en question.
- Messages de réponse multiples ou inattendus.

  L'importation n'attend qu'un seul message de réponse pour chaque message de demande. Egalement la période valide durant laquelle une réponse peut être reçue par le qualifiant d'expiration de réponse SCA sur la demande. Lorsqu'une réponse est reçue ou le délai d'expiration expire, l'enregistrement de corrélation

est supprimé. Si des messages de réponse parviennent de manière inattendue ou tardive, une exception Service Runtime est générée.

- Exception d'exécution lié au délai d'expiration du service, générée par une réponse tardive lors de l'utilisation du schéma de corrélation des destinations de réponse dynamique temporaires.

  L'importation JMS arrive à expiration après un délai déterminé par le qualifiant d'expiration des réponses SCA ; si ce délai n'est pas défini, il est de 60 secondes par défaut.

Messages SCA basés sur JM qui n'apparaissent pas dans le gestionnaire des événements ayant échoué

Si les messages SCA émis par le biais d'une interaction JMS échouent, vous devriez les retrouver dans le gestionnaire des événements ayant échoué. Or, si ces messages n'apparaissent pas, vérifiez que la destination SIB sous-jacente de la destination JMS possède une valeur du nombre maximal de livraisons ayant échoué supérieure à 1. Définir cette valeur sur 2 ou plus permet une interaction avec le gestionnaire des événements ayant échoué au cours des appels SCA pour les liaisons JMS.

Gestion des exceptions

La configuration de la liaison détermine la manière dont sont gérées les exceptions qui sont émises par les gestionnaires de données ou les liaisons de données. En outre, la nature du flux de médiation détermine le comportement du système lorsqu'une exception de ce type est générée.

Différents incidents peuvent se produire lorsqu'un gestionnaire de données ou une liaison de données est appelé(e) par votre liaison. Par exemple, un gestionnaire de données peut recevoir un message dont la charge est endommagée ou il peut essayer de lire un message dont le format n'est pas correct.

Le traitement d'une telle exception par votre liaison est déterminé par la manière dont vous implémentez le gestionnaire de données ou la liaison de données. Il est ainsi recommandé de configurer votre liaison de données afin qu'elle génère une exception de type `DataBindingException`.

Lorsqu'une exception d'exécution, y compris une exception `DataBindingException` est émise :

- Si le flux de médiation est configuré pour être transactionnel, le message JMS, par défaut, est stocké dans le gestionnaire des événements ayant échoué pour une relecture manuelle ou une suppression.

  Remarque : Vous pouvez modifier le mode de récupération sur la liaison afin que le message soit annulé plutôt que stocké dans le gestionnaire des événements ayant échoué.
- Si le flux de médiation n'est pas de type transactionnel, l'exception est consignée et le message est perdu.

La situation est similaire pour un gestionnaire de données. Dès lors que le gestionnaire de données est appelé par la liaison de données, toute exception de gestionnaire de données est encapsulée dans une exception de liaison de données. Ainsi, une exception `DataHandlerException` vous est signalée sous la forme d'une exception `DataBindingException`.

Liaisons JMS génériques

La liaison JMS générique assure la connectivité avec les fournisseurs tiers compatibles avec JMS 1.1. Le fonctionnement des liaisons JMS génériques est identique à celui des liaisons JMS.

Le service fourni par le biais d'une liaison JMS permet à un module SCA (Service Component Architecture) d'effectuer des appels ou de recevoir des messages à partir de systèmes externes. Le système peut être un système JMS externe.

La liaison JMS générique assure l'intégration avec les fournisseurs JMS conformes non JCA 1.5 qui prennent en charge JJMS 1.1 et implémentent la fonction de serveur d'applications JMS disponible en option. La liaison JMS générique prend en charge les fournisseurs JMS (notamment Oracle AQ, TIBCO, SonicMQ, WebMethods, BEA WebLogic et WebSphere MQ) qui ne prennent pas en charge JCA 1.5 mais qui prennent en charge la fonction de serveur d'applications de la spécification JMS 1.1. Le fournisseur JMS imbriqué dans WebSphere (SIBJMS), qui est un fournisseur JMS JCA 1.5, n'est pas pris en charge par cette liaison ; si vous utilisez ce fournisseur, utilisez les «Liaisons JMS», à la page 85.

Cette liaison générique peut par exemple être utilisée lors d'une intégration avec un système JMS conforme non JCA 1.5 dans un environnement. Les applications externes cibles peuvent alors recevoir et envoyer des messages à intégrer avec un composant SCA.

Concepts associés

«Présentation des liaisons JMS génériques»
Les liaisons JMS génériques sont des liaisons JMS non JCA qui assurent la connectivité entre l'environnement SCA (Service Component Architecture) et les systèmes JMS compatibles avec JMS 1.1 et qui implémentent la fonction de serveur d'applications JMS en option.

«En-têtes JMS génériques», à la page 101
Les en-têtes JMS génériques sont des objets SDO (Service Data Objects) qui contiennent toutes les propriétés des propriétés de message JMS génériques. Ces propriétés peuvent provenir du message entrant ou il peut s'agir des propriétés qui seront appliquées au message sortant.

«Résolution d'incidents liés aux liaisons JMS génériques», à la page 102
Vous pouvez diagnostiquer et résoudre des incidents survenant sur les liaisons JMS génériques.

«Gestion des exceptions», à la page 104
La configuration de la liaison détermine la manière dont sont gérées les exceptions qui sont émises par les gestionnaires de données ou les liaisons de données. En outre, la nature du flux de médiation détermine le comportement du système lorsqu'une exception de ce type est générée.

Référence associée

«Principales fonctionnalités des liaisons JMS Generic», à la page 100
Les fonctions de la liaison Generic d'importation et d'exportation JMS correspondent aux fonctions des liaisons d'importation JMS et MQ JMS intégrées à WebSphere. Les fonctions essentielles incluent les définitions d'en-tête et l'accès aux ressources Java EE existantes. Cependant, étant donné son caractère générique, cette liaison ne comporte pas d'options de connectivité spécifiques au fournisseur JMS et elle est limitée à la génération de ressources lors du déploiement et de l'installation.

Présentation des liaisons JMS génériques

Les liaisons JMS génériques sont des liaisons JMS non JCA qui assurent la connectivité entre l'environnement SCA (Service Component Architecture) et les systèmes JMS compatibles avec JMS 1.1 et qui implémentent la fonction de serveur d'applications JMS en option.

Liaisons JMS génériques

Les aspects majeurs des liaisons d'importation et d'exportation JMS génériques sont :

- Port de l'écouteur : permet aux fournisseurs JMS non JCA de recevoir des messages et de les distribuer à un bean géré par message
- Connexions : encapsulent une connexion virtuelle entre un client et une application fournisseur
- Destinations : utilisées par un client pour spécifier la cible des messages produits ou la source des messages utilisés
- Données d'authentification : permettent de sécuriser l'accès à la liaison

Liaisons d'importation JMS génériques

Les liaisons d'importation JMS génériques permettent aux composants au sein de votre module SCA de communiquer avec les services fournis par les fournisseurs JMS conformes non JCA 1.5.

La connexion qui fait partie d'une importation JMS est une fabrique de connexions. Une fabrique de connexion, l'objet utilisé par un client pour créer une connexion à un fournisseur, encapsule un ensemble de paramètres de configuration de la connexion définie par un administrateur. Chaque fabrique de connexions est une instance de l'interface ConnectionFactory, QueueConnectionFactory ou TopicConnectionFactory.

L'interaction avec les systèmes JMS externes comprend l'utilisation des destinations pour l'envoi des requêtes et la réception des réponses.

Deux types de scénarios d'utilisation pour la liaison d'import ation JMSgénérique sont pris en charge, en fonction du type d'opération appelée :

- Unidirectionnel : l'importation JMS générique place un message sur la destination d'envoi configurée dans la liaison d'importation. Rien n'est envoyé à la zone replyTo de l'en-tête JMS.
- Bidirectionnel (demande-réponse) : l'importation JMS générique place un message sur la destination d'envoi et conserve ensuite la réponse reçue depuis le composant SCA.

  La destination receive est définie dans la propriété d'en-tête replyTo du message sortant. Un bean géré par message (MDB) est déployé pour écouter sur la destination de réception et, dès qu'une réponse est reçue, le MDB transmet la réponse au composant.

  La liaison d'importation peut être configurée (à l'aide de la zone **Schéma de corrélation de réponse** dans WebSphere Integration Developer) pour faire en sorte que l'ID de corrélation de message de réponse soit copié à partir de l'ID de message de demande (valeur par défaut) ou à partir de l'ID de corrélation de message de demande.

Pour les scénarios d'utilisation unidirectionnel et bidirectionnel, les propriétés d'en-tête dynamique et statique peuvent être spécifiées. Les propriétés statiques peuvent être définies à partir de la liaison de méthode d'importation JMS générique. Certaines de ces propriétés ont une signification particulière pour l'environnement d'exécution JMS SCA.

Il est important de noter que la liaison JMS générique est une liaison asynchrone. Si un composant appelant appelle une importation JMS générique de manière synchrone (pour une opération bidirectionnelle), le composant appelant est bloqué jusqu'à ce que la réponse soit renvoyée par le service JMS.

La figure 36, à la page 99 illustre la manière dont l'importation est liée au service externe.

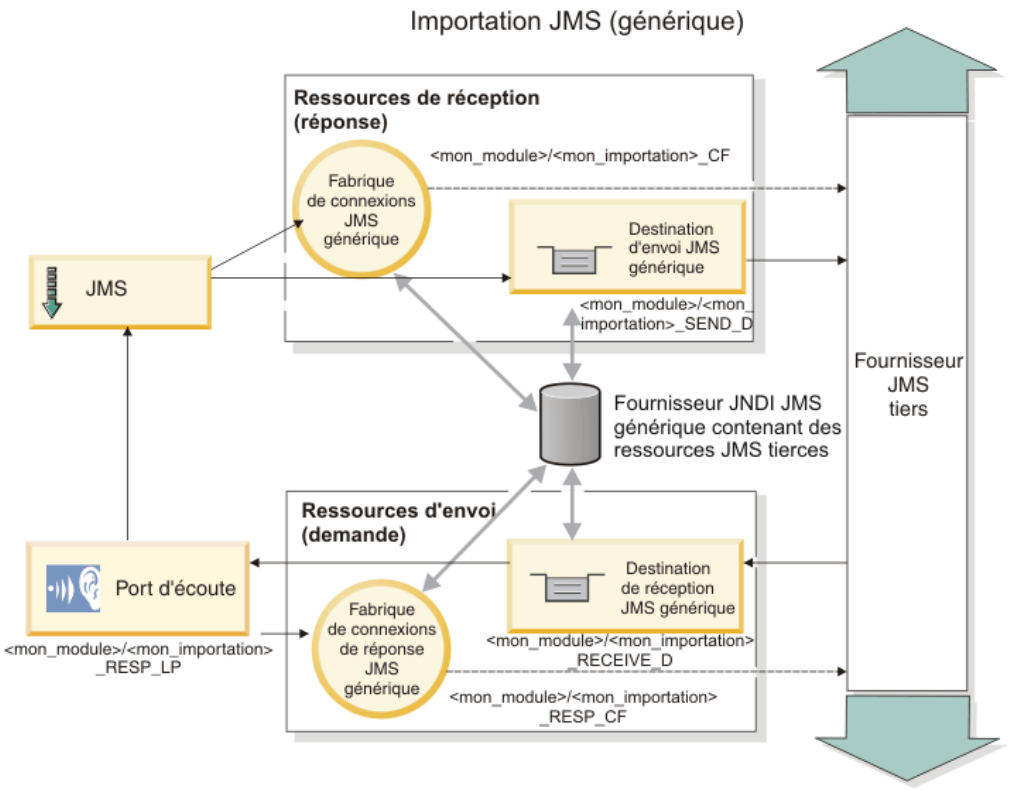


Figure 36. Ressources de liaisons d'importation JMS génériques

Liaisons d'exportation JMS génériques

Les liaisons d'exportation JMS génériques offrent les moyens aux modules SCA de fournir des services aux applications JMS externes.

La connexion qui fait partie d'une exportation JMS est composée d'une entité ConnectionFactory et d'une entité ListenerPort.

Une exportation JMS générique comporte des destinations d'envoi et de réception.

- La destination receive est le lieu de réception du message entrant destiné au composant cible.
- La destination send est celle à laquelle la réponse sera envoyée, saus si le message entrant l'a remplacé en utilisant la propriété d'en-tête replyTo.

Un bean MDB est déployé pour écouter les demandes parvenant à la destination receive spécifiée dans la liaison d'exportation.

- La destination spécifiée dans la zone send est utilisée pour envoyer la réponse à la demande entrante si le composant appelé fournit une réponse.
- La destination spécifiée dans la zone replyTo du message entrant remplace la destination spécifiée dans la zone send.
- Pour les scénarios de demande/réponse, la liaison d'importation peut être configurée (à l'aide de la zone **Schéma de corrélation de réponse** dans WebSphere Integration Developer) pour faire en sorte que la réponse copie l'ID de message de demande dans la zone ID de corrélation du message de réponse (option par défaut) ou la réponse peut copier l'ID de corrélation de la demande dans la zone ID de corrélation du message de réponse.

La figure 37 illustre la manière dont le demandeur externe est lié à l'exportation.

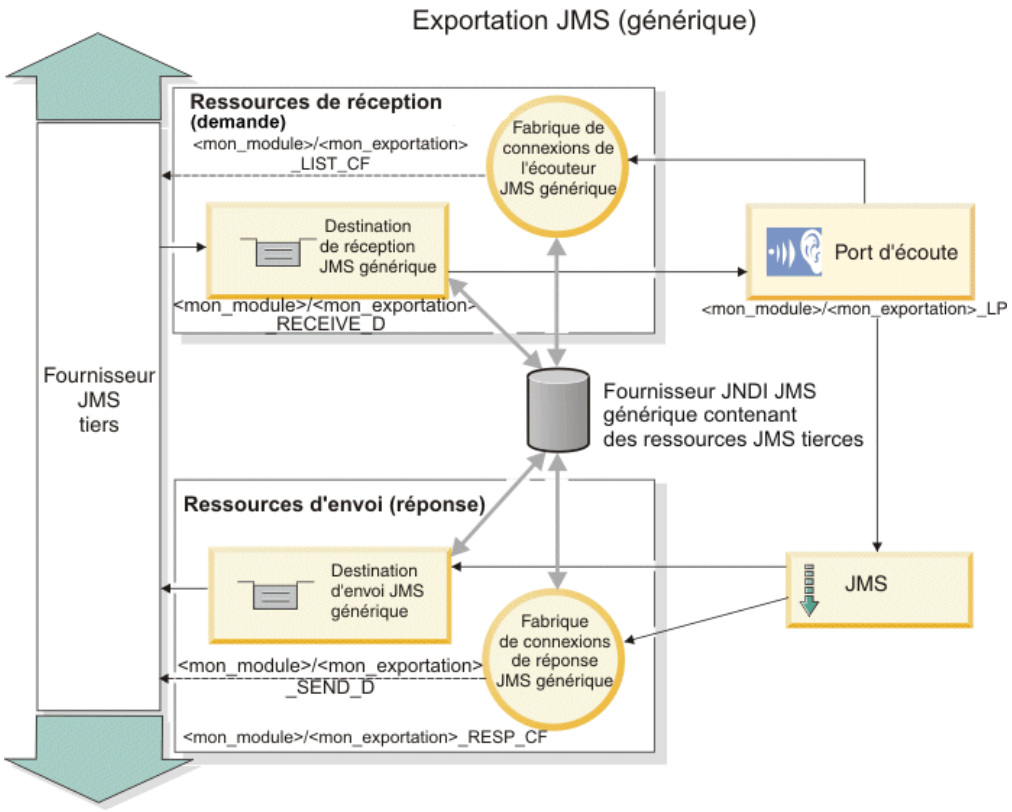


Figure 37. Ressources de liaisons d'exportation JMS génériques

Principales fonctionnalités des liaisons JMS Generic

Les fonctions de la liaison Generic d'importation et d'exportation JMS correspondent aux fonctions des liaisons d'importation JMS et MQ JMS intégrées à WebSphere. Les fonctions essentielles incluent les définitions d'en-tête et l'accès aux ressources Java EE existantes. Cependant, étant donné son caractère générique, cette liaison ne comporte pas d'options de connectivité spécifiques au fournisseur JMS et elle est limitée à la génération de ressources lors du déploiement et de l'installation.

Importations génériques

Comme l'application d'importation MQ JMS, l'implémentation JMS Generic est asynchrone et prend en charge trois types d'appel : unidirectionnel, bidirectionnel (également appelé demande-réponse) et rappel.

Si l'importation JMS est déployée, un bean géré par message (MDB) fourni par l'environnement d'exécution est déployé. Le bean MDB est à l'écoute des réponses au message de demande. Le bean MDB est associé à (écoute) la destination envoyée avec la demande dans la zone d'en-tête replyTo du message JMS.

Exportations génériques

Les liaisons d'exportation JMS génériques diffèrent des liaisons d'exportation EIS au niveau du traitement du renvoi du résultat. Une exportation JMS Generic

envoie explicitement la réponse à la destination replyTo spécifiée dans le message entrant. Si aucune destination n'est indiquée, la destination d'envoi est utilisée.

Lorsque l'exportation JMS Generic est déployée, un bean MDB (autre que le bean MDB utilisé pour les importations JMS Generic) est déployé. Il écoute les requêtes entrantes sur la destination de réception, puis distribue les requêtes que doit traiter l'environnement d'exécution SCA.

En-têtes spéciaux

Des propriétés d'en-tête spéciales sont utilisées dans les importations et les exportations JMS Generic pour indiquer à la cible comment traiter le message.

Par exemple, la propriété TargetFunctionName permet au sélecteur de fonction par défaut d'identifier le nom de l'opération sur l'interface d'exportation appelée.

Remarque : Il est possible de configurer la liaison d'importation pour que l'en-tête TargetFunctionName porte le nom de l'opération, dans chaque cas.

Ressources Java EE

Plusieurs ressources Java EE sont créées lors du déploiement d'une liaison JMS dans un environnement Java EE.

- Port d'écoute sur la destination de réception (réponse) (bidirectionnel uniquement) pour les importations, et sur la destination de réception (demande) pour les exportations
- Fabrique de connexions JMS générique pour outboundConnection (importation) et inboundConnection (exportation)
- Destination JMS générique pour les destinations d'envoi (importation) et de réception (exportation) (bidirectionnel uniquement)
- Fabrique de connexions JMS générique pour responseConnection (bidirectionnel uniquement et facultatif), sinon : outboundConnection est utilisé pour les importations et inboundConnection est utilisé pour les exportations)
- Destination JMS générique pour la destination de réception (importation) et d'envoi (exportation) (bidirectionnel uniquement)
- Destination JMS de rappel de fournisseur de messagerie par défaut utilisée pour l'accès à la destination de file d'attente de rappel SIB (bidirectionnel uniquement)
- Fabrique de connexions JMS de rappel de fournisseur de messagerie par défaut utilisée pour l'accès à la destination JMS de rappel (bidirectionnel uniquement)
- Destination de file d'attente de rappel SIB utilisée pour stocker les informations sur le message de demande à utiliser lors du traitement de la réponse (bidirectionnel uniquement)

La tâche d'installation crée ConnectionFactory, les trois destinations et ActivationSpec à partir des informations des fichiers d'importation et d'exportation.

En-têtes JMS génériques

Les en-têtes JMS génériques sont des objets SDO (Service Data Objects) qui contiennent toutes les propriétés des propriétés de message JMS génériques. Ces propriétés peuvent provenir du message entrant ou il peut s'agir des propriétés qui seront appliquées au message sortant.

Un message JMS contient deux types d'en-têtes – l'en-tête de système JMS et plusieurs propriétés JMS. Il est possible d'accéder aux deux types d'en-tête depuis un module de médiation dans l'objet SMO (Service Message Object) ou en utilisant l'API ContextService.

Les propriétés suivantes sont définies statiquement sur methodBinding :

- JMSType
- JMSCorrelationID
- JMSDeliveryMode
- JMSPriority

La liaison JMS générique JMS prend également en charge la modification dynamique des propriétés et des en-têtes JMS de la même manière que les liaisons JMS et JMS MQ.

Certains fournisseurs JMS génériques instaurent des restrictions sur quelles propriétés pouvant être définies par l'application et avec quelles combinaisons. Pour plus d'informations, reportez-vous à la documentation de votre produit tiers. Toutefois, une propriété complémentaire a été ajoutée à methodBinding, ignoreInvalidOutboundJMSProperties, qui permet de propager n'importe quelle exception.

Les propriétés de message et d'en-tête JMS génériques sont uniquement utilisées lorsque le commutateur de liaisons SCDL de l'architecture de composants de service de base est sous tension. Lorsque le commutateur est activé, les informations contextuelles sont propagées. Par défaut, ce commutateur est sous tension. Pour éviter la diffusion des informations contextuelles, remplacez la valeur par `false`.

Lorsque la diffusion du contexte est activée, les informations d'en-tête peuvent être transmises au message ou au composant cible. Pour activer et désactiver la diffusion du contexte, spécifiez la valeur `true` ou `false` pour l'attribut contextPropagationEnabled des liaisons d'importation et d'exportation. Exemple :

```
<esbBinding xsi:type="eis:JMSImportBinding" contextProgagationEnabled="true">
```

La valeur par défaut est `true`.

Résolution d'incidents liés aux liaisons JMS génériques

Vous pouvez diagnostiquer et résoudre des incidents survenant sur les liaisons JMS génériques.

Exceptions liées à l'implémentation

En réponse à diverses conditions d'erreur, l'implémentation de l'importation et de l'exportation JMS générique peut renvoyer deux types d'exception :

- Service : cette exception est renvoyée si l'erreur définie sur l'interface métier de service (type de port WSDL) s'est produite.
- Service Runtime : générée dans tous les autres cas. Dans la plupart des cas, l'exception cause contient l'exception d'origine (JMSException).

Résolution d'une expiration de message JMS générique

Expiration d'un message de demande du fournisseur JMS.

L'*expiration de la demande* désigne l'expiration d'un message de demande du fournisseur JMS à la fin du délai JMSExpiration indiqué dans le message de demande. Comme pour d'autres liaisons JMS, la liaison JMS générique traite l'expiration de la demande en attribuant à l'expiration du message de rappel placé lors de l'importation la même valeur que celle de la demande sortante. La notification de l'expiration du message de rappel indique que le message de demande a expiré et le client doit être informé au moyen d'une exception.

Cependant, si la destination de rappel est transférée sur le fournisseur tiers, ce type d'expiration de demande n'est pas pris en charge.

L'*expiration de la réponse* désigne l'expiration d'un message de réponse du fournisseur JMS à la fin du délai JMSExpiration indiqué dans le message de réponse.

L'expiration de la réponse n'est pas prise en charge pour la liaison JMS générique, car le fonctionnement d'une expiration sur un fournisseur JMS tiers n'est pas définie. Toutefois, vous pouvez déterminer si la réponse a expiré lors de sa réception.

Pour les messages de demande sortants, la valeur de JMSExpiration est calculée à partir du temps d'attente et des valeurs requestExpiration indiquées dans asyncHeader, si elles sont définies.

Résolution d'erreurs liées aux fabriques de connexion JMS génériques

Lorsque vous définissez certains types de fabriques de connexion au niveau du fournisseur JMS générique, vous pouvez recevoir un message d'erreur lors d'une tentative de lancement d'une application. Vous pouvez modifier la fabrique de connexions externe pour empêcher ce problème de se produire.

Lorsque vous lancez une application, le message suivant peut s'afficher :

```
Le type JMSConnectionFactory du port d'écoute MDB ne correspond pas
au type JMSDestination
```

Cet incident peut survenir lorsque vous définissez des fabriques de connexions externes. En particulier, l'exception peut être générée lorsque vous créez une fabrique de connexions de sujet JMS 1.0.2 au lieu d'une fabrique de connexions JMS 1.1 (unifiée) (c'est-à-dire, une fabrique qui prenne en charge les communications de type point à point et publication/abonnement).

Pour résoudre ce problème, procédez comme suit :

1. Accédez au fournisseur JMS générique que vous utilisez.
2. Remplacez la fabrique de connexions de sujet JMS 1.0.2 que vous avez définie par une fabrique de connexions JMS 1.1 (unifiée).

Lorsque vous lancez l'application avec la nouvelle fabrique de connexions JMS 1.1, le message d'erreur ne devrait plus s'afficher.

Messages SCA génériques basés sur JMS qui n'apparaissent pas dans le gestionnaire des événements ayant échoué

Si les messages SCA émis par le biais d'une interaction JMS générique échouent, vous devriez les retrouver dans le gestionnaire des événements ayant échoué. Si ces messages n'apparaissent pas, vérifiez que la valeur de la propriété du nombre

maximal de nouvelles tentatives sur le port d'écoute sous-jacent est supérieure ou égale à 1. Définir cette valeur sur 1 ou plus permet une interaction avec le gestionnaire d'événements ayant échoué au cours des appels SCA pour les liaisons JMS génériques.

Gestion des exceptions

La configuration de la liaison détermine la manière dont sont gérées les exceptions qui sont émises par les gestionnaires de données ou les liaisons de données. En outre, la nature du flux de médiation détermine le comportement du système lorsqu'une exception de ce type est générée.

Différents incidents peuvent se produire lorsqu'un gestionnaire de données ou une liaison de données est appelé(e) par votre liaison. Par exemple, un gestionnaire de données peut recevoir un message dont la charge est endommagée ou il peut essayer de lire un message dont le format n'est pas correct.

Le traitement d'une telle exception par votre liaison est déterminé par la manière dont vous implémentez le gestionnaire de données ou la liaison de données. Il est ainsi recommandé de configurer votre liaison de données afin qu'elle génère une exception de type `DataBindingException`.

La situation est similaire pour un gestionnaire de données. Dès lors que le gestionnaire de données est appelé par la liaison de données, toute exception de gestionnaire de données est encapsulée dans une exception de liaison de données. Ainsi, une exception `DataHandlerException` vous est signalée sous la forme d'une exception `DataBindingException`.

Lorsqu'une exception d'exécution, y compris une exception `DataBindingException` est émise :

- Si le flux de médiation est configuré pour être transactionnel, le message JMS est stocké dans le gestionnaire des événements ayant échoué par défaut pour une relecture manuelle ou une suppression.

  Remarque : Vous pouvez modifier le mode de récupération sur la liaison afin que le message soit annulé plutôt que stocké dans le gestionnaire des événements ayant échoué.
- Si le flux de médiation n'est pas de type transactionnel, l'exception est consignée et le message est perdu.

La situation est similaire pour un gestionnaire de données. Le gestionnaire de données étant appelé par la liaison de données, une exception de gestionnaire de données est générée dans une exception de liaison de données. Par conséquent, une exception `DataHandlerException` vous est signalée sous la forme d'une exception `DataBindingException`.

Liaisons JMS WebSphere MQ

La liaison JMS WebSphere MQ assure l'intégration avec les applications externes qui utilisent un fournisseur basé sur JMS WebSphere MQ.

Utilisez les liaisons d'importation et d'exportationJMS WebSphere MQ lorsque vous souhaitez une intégration directe avec des systèmes JMS ou JMS MQ à partir de votre environnement de serveur. Ainsi, il n'est plus nécessaire d'utiliser les fonctions de lien MQ ou client du bus d'intégration de services.

Lorsqu'un composant interagit avec un service basé sur JMS WebSphere MQ par le biais d'une importation, la liaison d'importation JMS WebSphere MQ utilise une

destination vers laquelle les données seront envoyées et une destination à laquelle la réponse peut être reçue. La conversion des données vers ou depuis un message JMS est accomplie via le composant Edge Component du gestionnaire de données ou de la liaison de données.

Lorsqu'un module SCA fournit un service aux clients JMS WebSphere MQ, la liaison d'exportation JMS WebSphere MQ utilise une destination à laquelle la requête peut être reçue et la réponse envoyée. La conversion des données vers et depuis un message JMS s'effectue par le biais du gestionnaire de données ou de la liaison de données JMS.

Le sélecteur de fonction sert à effectuer un mappage avec l'opération sur le composant cible à appeler.

Concepts associés

«Présentation des liaisons JMS WebSphere MQ»
La liaison JMS WebSphere MQ assure l'intégration avec les applications externes qui utilisent le fournisseur JMS WebSphere MQ.

«En-têtes JMS», à la page 90
Un message JMS contient deux types d'en-têtes – l'en-tête de système JMS et plusieurs propriétés JMS. Il est possible d'accéder aux deux types d'en-tête depuis un module de médiation dans l'objet SMO (Service Message Object) ou en utilisant l'API ContextService.

«Clients externes», à la page 111
Le serveur peut échanger des messages (envoi et réception) avec des clients externes par le biais de liaisons JMS WebSphere MQ.

«Identification et résolution des incidents liés aux liaisons JMS WebSphere MQ», à la page 111
Vous pouvez diagnostiquer et résoudre des incidents survenant sur les liaisons JMS WebSphere MQ.

«Gestion des exceptions», à la page 104
La configuration de la liaison détermine la manière dont sont gérées les exceptions qui sont émises par les gestionnaires de données ou les liaisons de données. En outre, la nature du flux de médiation détermine le comportement du système lorsqu'une exception de ce type est générée.

Référence associée

«Principales fonctionnalités des liaisons JMS WebSphere MQ», à la page 108
Les fonctionnalités essentielles des liaisons JMS WebSphere MQ incluent les en-têtes, les artefacts Java EE et les ressources Java EE créées.

Présentation des liaisons JMS WebSphere MQ

La liaison JMS WebSphere MQ assure l'intégration avec les applications externes qui utilisent le fournisseur JMS WebSphere MQ.

Tâches d'administration WebSphere MQ

Avant d'exécuter une application contenant ce type de liaison, l'administrateur système WebSphere MQ est censé créer le gestionnaire de files d'attente WebSphere MQ sous jacent qui sera utilisé par les liaisons JMS WebSphere MQ.

Liaisons d'importation JMS WebSphere MQ

L'importation JMS WebSphere MQ permet aux composants au sein de votre module SCA de communiquer avec des services offerts par les fournisseurs basés sur JMS WebSphere MQ. Vous devez utiliser une version prise en charge de

WebSphere MQ. Pour plus de détails sur la configuration logicielle et matérielle requise, reportez-vous aux pages de support IBM..

Deux types de scénarios d'utilisation pour les liaisons d'importation JMS WebSphere MQ Jsont pris en charge, en fonction du type d'opération appelée :

- Unidirectionnel : l'importation JMS WebSphere MQ place un message sur la destination d'envoi configurée dans la liaison d'importation. Rien n'est envoyé à la zone replyTo de l'en-tête JMS.
- Bidirectionnel (demande-réponse) : l'importation JMS WebSphere MQ place un message sur la destination d'envoi.

  La destination receive est définie dans la zone d'en-tête replyTo. Un bean géré par message (MDB) est déployé pour écouter sur la destination de réception et, dès qu'une réponse est reçue, le MDB transmet la réponse au composant.

  La liaison d'importation peut être configurée (à l'aide de la zone **Schéma de corrélation de réponse** dans WebSphere Integration Developer) pour faire en sorte que l'ID de corrélation de message de réponse soit copié à partir de l'ID de message de demande (valeur par défaut) ou à partir de l'ID de corrélation de message de demande.

Pour les scénarios d'utilisation unidirectionnels et bidirectionnels, les propriétés d'en-tête dynamiques et statiques peuvent être spécifiées.Les propriétés statiques peuvent être définies à partir de la liaison de méthode d'importation JMS. Certaines de ces propriétés revêtent des significations particulières pour l'environnement d'exécution JMS SCA.

Il est important de noter que JMS WebSphere MQ est une liaison asynchrone. Si un composant appelant appelle une importation JMS WebSphere MQ de manière synchrone (pour une opération bidirectionnelle), le composant appelant est bloqué jusqu'à ce que la réponse soit renvoyée par le service.

La figure 38, à la page 107 montre comment l'importation est liée au service externe.

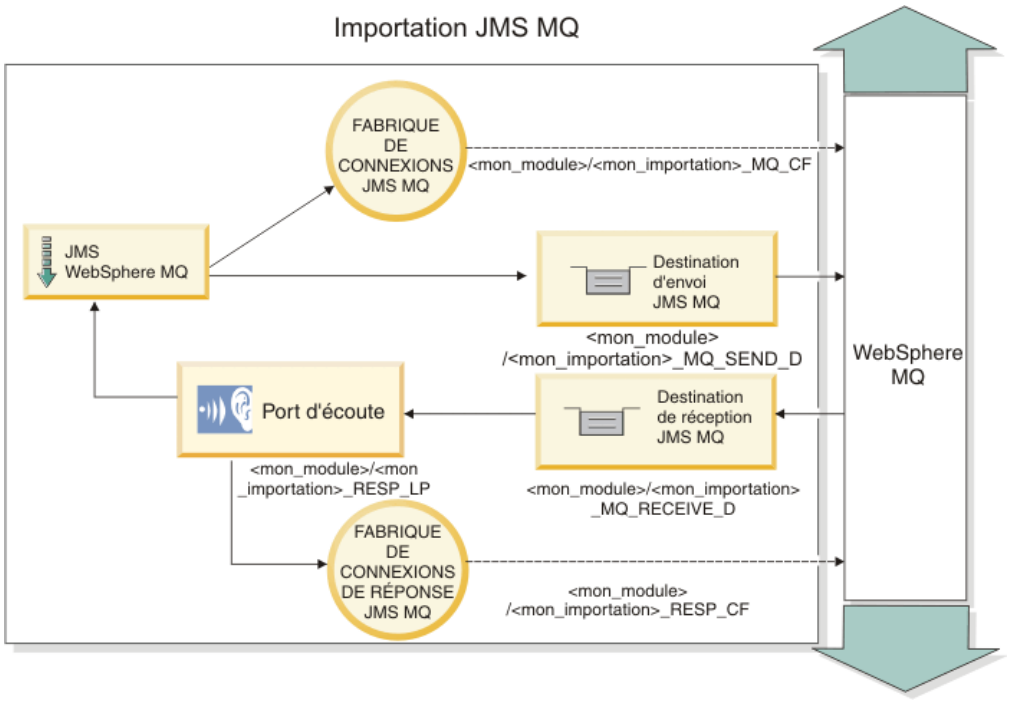


Figure 38. Ressources de liaison d'importation JMS WebSphere MQ

Liaisons d'exportation JMS Liaisons JMS WebSphere MQ

La liaison d'exportation JMS WebSphere MQ offre les moyens aux modules SCA de fournir des services aux applications JMS externes sur le fournisseur JMS basé sur WebSphere MQ.

Un bean MDB est déployé pour écouter les demandes parvenant à la destination receive spécifiée dans la liaison d'exportation. La destination spécifiée dans la zone send est utilisée pour envoyer la réponse à la demande entrante si le composant appelé fournit une réponse. La destination spécifiée dans la zone replyTo du message de réponse remplace la destination spécifiée dans la zone send.

La figure 39, à la page 108 montre comment le demandeur externe est lié à l'exportation.

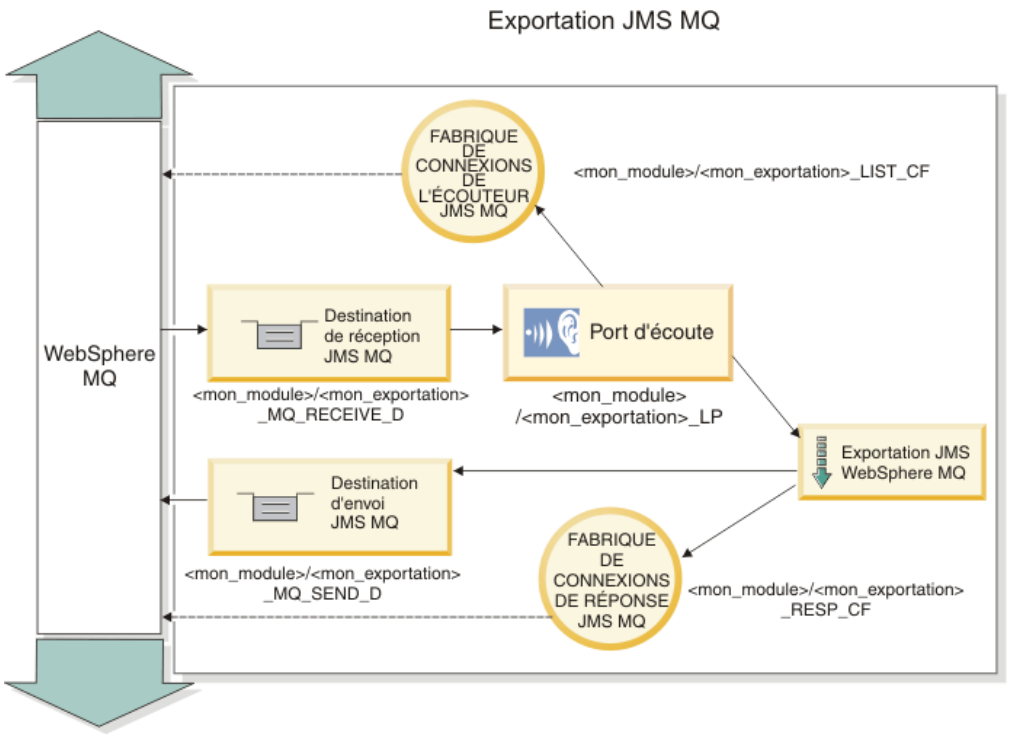


Figure 39. Ressources de liaison d'exportation JMS WebSphere MQ

Remarque : La figure 38, à la page 107 et la figure 39 illustrent comment une application d'une version précédente de WebSphere Process Server est associée à un service externe. Pour les applications développées pour WebSphere Process Server, version 7.0, la spécification d'activation est utilisée à la place du port de l'écouteur et de la fabrique de connexions.

Principales fonctionnalités des liaisons JMS WebSphere MQ

Les fonctionnalités essentielles des liaisons JMS WebSphere MQ incluent les en-têtes, les artefacts Java EE et les ressources Java EE créées.

En-têtes

Un en-tête de message JMS contient plusieurs zones prédéfinies qui contiennent des valeurs utilisées par les clients et les fournisseurs pour identifier et acheminer les messages. Vous pouvez utiliser les propriétés de liaison pour configurer ces en-têtes avec des valeurs fixes, ou vous pouvez spécifier dynamiquement les en-têtes lors de l'exécution.

JMSCorrelationID
Liaison avec un message associé. En général, cette zone est définie sur la chaîne de l'identificateur du message auquel une réponse est envoyée.

TargetFunctionName
Cet en-tête est utilisé par l'un des sélecteurs de fonction fournis pour identifier l'opération appelée. La définition de la propriété d'en-tête JMS TargetFunctionName dans des messages envoyés à une exportation JMS permet l'utilisation de ce sélecteur de fonction. La propriété peut être définie directement dans les applications client JMS ou lors de la connexion d'une importation avec une liaison JMS à une exportation de ce type. Dans ce cas,

vous devez configurer la liaison d'importation JMS pour que l'en-tête TargetFunctionName pour chaque opération de l'interface soit défini sur le nom de l'opération.

Schémas de corrélation

Les liaisons JMS WebSphere MQ fournissent un grand nombre de schémas de corrélation qui permettent de déterminer la manière de corréler les messages de demande avec les messages de réponse.

RequestMsgIDToCorrelID
JMSMessageID est copié dans la zone JMSCorrelationID. Il s'agit du paramètre par défaut.

RequestCorrelIDToCorrelID
JMSCorrelationID est copié dans la zone JMSCorrelationID.

Ressources Java EE

Plusieurs ressources Java EE sont créées lors du déploiement d'une importation JMS MQ dans un environnement Java EE.

Paramètres

Fabrique de connexions MQ
Utilisée par les clients pour créer une connexion au fournisseur JMS MQ.

Fabrique de connexions de réponse
Utilisée par l'environnement d'exécution JMS MQ SCA lorsque la destination d'envoi se trouve sur un gestionnaire de files d'attente différent de celui de la destination d'envoi.

Spécification d'activation
Une spécification d'activation MQ JMS est associée à un ou plusieurs beans gérés par message et offre la configuration permettant de recevoir des messages.

Destinations

- Destination d'envoi :
  - Importations : Destination à laquelle la demande ou le message sortant est envoyé.
  - Exportations : Emplacement auquel est envoyé le message de réponse, si cette valeur n'est pas remplacée par l'en-tête JMSReplyTo dans le message entrant.
- Destination de réception :
  - Importations : Emplacement auquel est envoyé le message de réponse ou entrant.
  - Exportations : Emplacement dans lequel le message entrant doit être placé.

En-têtes JMS

Un message JMS contient deux types d'en-têtes – l'en-tête de système JMS et plusieurs propriétés JMS. Il est possible d'accéder aux deux types d'en-tête depuis un module de médiation dans l'objet SMO (Service Message Object) ou en utilisant l'API ContextService.

En-tête système JMS

L'en-tête système JMS est représenté dans l'objet SMO par l'élément JMSHeader qui contient toutes les zones généralement présentes dans un en-tête JMS. Bien qu'elles puissent être modifiées dans la médiation (ou ContextService), certaines zones d'en-tête système JMS définies dans l'objet SMO ne seront pas propagées dans le message JMS sortant puisqu'elles sont remplacées par des valeurs statiques ou système.

Les zones clés de l'en-tête système JMS qui peuvent être mises à jour dans une médiation (ou ContextService) sont :

- **JMSType** et **JMSCorrelationID** : valeurs des propriétés de l'en-tête du message prédéfinies spécifiques
- **JMSDeliveryMode** – valeurs du mode de livraison (persistant (par défaut) ou non persistant)
- **JMSPriority** : valeur de la priorité (0 à 9 ; la valeur par défaut est JMS_Default_Priority)

Propriétés JMS

Les propriétés JMS sont représentées dans l'objet SMO en tant qu'entrées dans la liste Propriétés. Les propriétés peuvent être ajoutées, mises à jour ou supprimées dans une médiation ou en utilisant l'API ContextService.

Elles peuvent également être définies de manière statique dans la liaison JMS. Les propriétés définies de manière statique remplacent les paramètres (portant le même nom) qui sont définis de manière dynamique.

Les propriétés utilisateur propagées à partir d'autres liaisons (par exemple, une liaison HTTP) correspondront à une sortie dans la liaison JMS comme les propriétés JMS.

Paramètres de propagation d'en-tête

La propagation des propriétés et des en-têtes système JMS depuis le message JMS entran t vers les composants en aval ou depuis les composants en amont vers le message JMS sortant peut être contrôlée par l'indicateur Propagate Protocol Header.

Lorsque Propagate Protocol Header est défini, les informations d'en-tête peuvent circuler vers le message ou vers le composant cible, comme indiqué dans la liste suivante :

- Requête d'exportation JMS

  L'en-tête JMS reçu dans le message sera propagé aux composants cible via le service de contexte. Les propriétés JMS reçues dans le message seront propagées aux composants cible via le service de contexte.
- Réponse d'exportation JMS

  Toute zone d'en-tête JMS définie dans le service de contexte sera utilisée dans le message sortant, excepté en cas de remplacement par les propriétés statiques définies sur la liaison d'exportation JMS. Toute propriété définie dans le service de contexte sera utilisée dans le message sortant excepté en cas de remplacement par les propriétés statiques définies sur la liaison d'exportation JMS.
- Requête d'importation JMS

Toute zone d'en-tête JMS définie dans le service de contexte sera utilisée dans le message sortant, excepté en cas de remplacement par les propriétés statiques définies sur la liaison d'importation JMS. Toute propriété définie dans le service de contexte sera utilisée dans le message sortant excepté en cas de remplacement par les propriétés statiques définies sur la liaison d'importation JMS.

- Réponse d'importation JMS

  L'en-tête JMS reçu dans le message sera propagé aux composants cible via le service de contexte. Les propriétés JMS reçues dans le message seront propagées aux composants cible via le service de contexte.

Clients externes

Le serveur peut échanger des messages (envoi et réception) avec des clients externes par le biais de liaisons JMS WebSphere MQ.

Un client externe (comme un portail Web ou un système d'information d'entreprise) peut envoyer un message à un composant SCA dans l'application par le biais d'une exportation ou il peut être appelé par un composant SCA dans l'application via une importation.

La liaison d'exportation JMS WebSphere MQ déploie les beans MDB (Message Driven Beans) pour écouter les demandes entrantes sur la destination receive spécifiée dans la liaison d'exportation. La destination spécifiée dans la zone send est utilisée pour envoyer la réponse à la demande entrante si l'application appelée fournit une réponse. Ainsi, un client externe est en mesure d'appeler des applications via la liaison d'exportation.

Les importations JMS WebSphere MQ JMS effectuent des liaisons avec des clients externes et peuvent ainsi leur envoyer des messages. Ces messages peuvent ou non exiger une réponse de la part du client externe.

Pour plus d'informations sur l'interaction avec des clients externes à l'aide de WebSphere MQ, reportez-vous au centre d'information WebSphere MQ.

Identification et résolution des incidents liés aux liaisons JMS WebSphere MQ

Vous pouvez diagnostiquer et résoudre des incidents survenant sur les liaisons JMS WebSphere MQ.

Exceptions liées à l'implémentation

En réponse à diverses conditions d'erreur, l'implémentation de l'importation et de l'exportation JMS MQ peut renvoyer deux types d'exception :

- Service : cette exception est renvoyée si l'erreur définie sur l'interface métier de service (type de port WSDL) s'est produite.
- Service Runtime : générée dans tous les autres cas. Dans la plupart des cas, l'exception cause contient l'exception d'origine (JMSException).

  Par exemple, une importation n'attend qu'un seul message de réponse pour chaque message de demande. Si plusieurs réponses sont reçues, ou si une réponse tardive (une pour laquelle la réponse SCA a expiré) est reçue, une exception Service Runtime est générée. La transaction est annulée et le message de réponse est retiré de la file d'attente ou traité par le gestionnaire d'événements ayant échoué.

Messages SCA basés sur JMS WebSphere MQ qui n'apparaissent pas dans le gestionnaire des événements ayant échoué

Si les messages SCA émis par le biais d'une interaction JMS WebSphere MQ échouent, vous devriez les retrouver dans le gestionnaire des événements ayant échoué. Or, si ces messages n'apparaissent pas, vérifiez que la valeur de la propriété du nombre maximal de nouvelles tentatives sur le port d'écoute sous-jacent est supérieure ou égale à 1. Définir cette valeur sur 1 ou plus permet une interaction avec le gestionnaire d'événements ayant échoué au cours des appels SCA pour les liaisons JMS MQ.

Scénarios d'utilisation incorrecte : comparaison avec les liaisons WebSphere MQ

La liaison JMS WebSphere MQ est conçue pour l'interopérabilité avec les applications JMS déployées sur WebSphere MQ, où les messages affichés reposent sur un modèle de message JMS. Cependant, l'importation et l'exportationWebSphere MQ sont conçues essentiellement pour interopérer avec les applications WebSphere MQ natives et exposer le contenu intégral du corps de message WebSphere MQ aux médiations.

Dans les scénarios suivants, il est nécessaire d'utiliser la liaison JMSWebSphere MQ et non la liaison WebSphere MQ :

- Appel d'un bean géré par message (MDB) JMS depuis un module SCA, où le MDB est déployé sur le fournisseur JMS WebSphere MQ. Utilisez une importation JMS WebSphere MQ.
- Permettre au module SCA d'être appelé à partir d'un composant de servlet Java EE ou EJB par JMS. Utilisez une exportation JMS WebSphere MQ.
- Médiation du contenu d'un JMS MapMessage, transitant dans WebSphere MQ. Utilisez une exportation et une importation JMS WebSphere MQ conjointement avec le gestionnaire de données ou la liaison de données appropriés.

Dans certaines situations, la liaison WebSphere MQ et la liaison JMSWebSphere MQ peuvent interopérer. En particulier, si vous reliez des applications WebSphere MQ Java EE et non Java EE, utilisez une exportation WebSphere MQ et une importation JMS WebSphere MQ (ou vice-versa) conjointement avec les liaisons de données et/ou les modules de médiation appropriés (ou les deux).

Gestion des exceptions

La configuration de la liaison détermine la manière dont sont gérées les exceptions qui sont émises par les gestionnaires de données ou les liaisons de données. En outre, la nature du flux de médiation détermine le comportement du système lorsqu'une exception de ce type est générée.

Différents incidents peuvent se produire lorsqu'un gestionnaire de données ou une liaison de données est appelé(e) par votre liaison. Par exemple, un gestionnaire de données peut recevoir un message dont la charge est endommagée ou il peut essayer de lire un message dont le format n'est pas correct.

Le traitement d'une telle exception par votre liaison est déterminé par la manière dont vous implémentez le gestionnaire de données ou la liaison de données. Il est ainsi recommandé de configurer votre liaison de données afin qu'elle génère une exception de type `DataBindingException`.

La situation est similaire pour un gestionnaire de données. Dès lors que le gestionnaire de données est appelé par la liaison de données, toute exception de gestionnaire de données est encapsulée dans une exception de liaison de données. Ainsi, une exception `DataHandlerException` vous est signalée sous la forme d'une exception `DataBindingException`.

Lorsqu'une exception d'exécution, y compris une exception `DataBindingException` est émise :

- Si le flux de médiation est configuré pour être transactionnel, le message JMS est stocké dans le gestionnaire des événements ayant échoué par défaut pour une relecture manuelle ou une suppression.

  Remarque : Vous pouvez modifier le mode de récupération sur la liaison afin que le message soit annulé plutôt que stocké dans le gestionnaire des événements ayant échoué.
- Si le flux de médiation n'est pas de type transactionnel, l'exception est consignée et le message est perdu.

La situation est similaire pour un gestionnaire de données. Le gestionnaire de données étant appelé par la liaison de données, une exception de gestionnaire de données est générée dans une exception de liaison de données. Par conséquent, une exception `DataHandlerException` vous est signalée sous la forme d'une exception `DataBindingException`.

Liaisons WebSphere MQ

La liaison WebSphere MQ assure une connectivité SCA (Service Component Architecture) avec les applications WebSphere MQ.

Utilisez les liaisons d'importation et d'exportation WebSphere MQ pour une intégration directe avec le système basé sur WebSphere MQ à partir de votre environnement serveur. Ainsi, il n'est plus nécessaire d'utiliser les fonctions de lien MQ ou client du bus d'intégration de services.

Lorsqu'un composant interagit avec un service WebSphere MQ par le biais d'une importation, la liaison d'importation WebSphere MQ utilise unefile d'attente vers laquelle les données seront envoyées et une file d'attente où la réponse peut être reçue.

Lorsqu'un module SCA fournit un service aux clients WebSphere MQ, la liaison d'exportation WebSphere MQ utilise une file d'attente où la requête peut être reçue et la réponse envoyée. Le sélecteur de fonction sert à effectuer un mappage avec l'opération sur le composant cible à appeler.

La conversion des données utiles vers ou depuis un message MQ est effectuée par l'intermédiaire de la liaison de données ou du gestionnaire de données de corps MQ. La conversion des données d'en-tête vers et depuis un message MQ est effectuée par l'intermédiaire de la liaison de données d'en-tête MQ.

Pour plus d'informations sur les versions WebSphere MQ prises en charge, voir le site Web WebSphere Process Server system requirements.

Concepts associés

«Présentation des liaisons WebSphere MQ»
La liaison WebSphere MQ assure l'intégration avec les applications natives basées sur MQ.

«En-têtes WebSphere MQ», à la page 118
Les en-têtes WebSphere MQ incorporent certaines conventions pour la conversion aux messages de l'architecture SCA (Service Component Architecture).

«Ajout statique de MQCIH dans une liaison WebSphere MQ», à la page 120
WebSphere Process Server prend en charge l'ajout des informations d'en-tête MQCIH statiquement sans utiliser de module de médiation.

«Clients externes», à la page 121
WebSphere Process Server peut échanger des messages (envoi et réception) avec des clients externes par le biais de liaisons WebSphere MQ.

«Identification des incidents liés aux liaisons WebSphere MQ», à la page 121
Diagnostic et correction des incidents et des erreurs liés aux liaisons WebSphere MQ.

«Gestion des exceptions», à la page 104
La configuration de la liaison détermine la manière dont sont gérées les exceptions qui sont émises par les gestionnaires de données ou les liaisons de données. En outre, la nature du flux de médiation détermine le comportement du système lorsqu'une exception de ce type est générée.

Référence associée

«Principales fonctionnalités d'une liaison WebSphere MQ», à la page 116
Les fonctionnalités essentielles d'une liaison WebSphere MQ incluent les en-têtes, les artefacts Java EE et les ressources Java EE créées.

Présentation des liaisons WebSphere MQ

La liaison WebSphere MQ assure l'intégration avec les applications natives basées sur MQ.

Tâches d'administration WebSphere MQ

Avant d'exécuter une application contenant ce type de liaison, l'administrateur système WebSphere MQ est censé créer le gestionnaire de files d'attente WebSphere MQ sous jacent qui sera utilisé par les liaisons WebSphere MQ.

Tâches d'administration WebSphere

Vous devez spécifier comme propriété **Chemin d'accès aux bibliothèques natives**, de l'adaptateur de ressources MQ de Websphere, la version WebSphere MQ prise en charge par le serveur, puis redémarrer ce dernier. Cela garantit l'utilisation de bibliothèques d'une version de WebSphere MQ prise en charge. Pour plus de détails sur la configuration logicielle et matérielle requise, reportez-vous aux pages de support IBM

Liaisons d'importation WebSphere MQ

La liaison d'importation WebSphere MQ permet aux composants au sein de votre module SCA de communiquer avec des services offerts par des applications externes basées sur WebSphere MQ. Vous devez utiliser une version prise en charge de WebSphere MQ. Pour plus de détails sur la configuration logicielle et matérielle requise, reportez-vous aux pages de support IBM..

L'interaction avec les systèmes WebSphere MQ externes comprend l'utilisation de file d'attentes pour l'envoi des requêtes et la réception des réponses.

Deux types descénarios d'utilisation pour la liaison d'importation WebSphere MQ sont pris en charge, en fonction du type d'opération appelée :

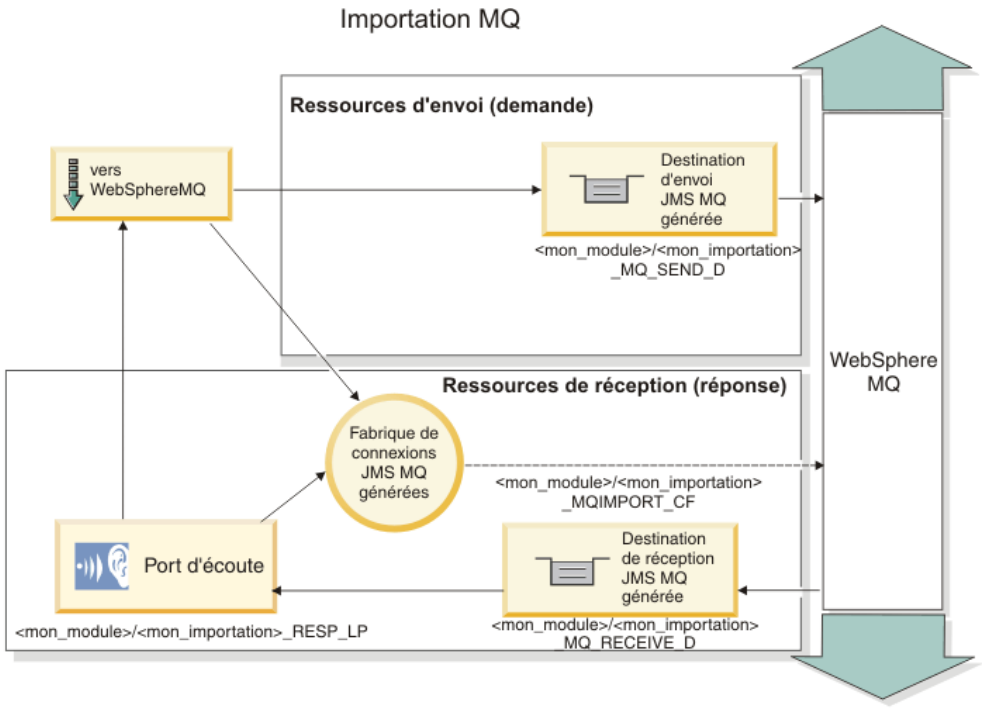
- Unidirectionnel : l'importation WebSphere MQ place un message sur la file d'attente configurée dans la zone **file d'attente de destination d'envoi** de la liaison d'importation. Rien n'est envoyé à la zone replyTo de l'en-tête MQMD.
- Bidirectionnel (demande-réponse) : l'importation WebSphere MQ place un message sur la file d'attente configurée dans la zone **file d'attente de destination d'envoi**

  La file d'attente receive est définie dans la zone d'en-tête MQMD replyTo. Un bean géré par message (MDB) est déployé pour écouter sur la file d'attente de réception et, dès qu'une réponse est reçue, le MDB transmet la réponse au composant.

  La liaison d'importation peut être configurée (à l'aide de la zone **Schéma de corrélation de réponse**) pour faire en sorte que l'ID de corrélation de message de réponse soit copié à partir de l'ID de message de demande (valeur par défaut) ou à partir de l'ID de corrélation de message de demande.

Il est important de noter que WebSphere MQ est une liaison asynchrone. Si un composant appelant appelle une importation WebSphere MQ de manière synchrone (pour une opération bidirectionnelle), le composant appelant est bloqué jusqu'à ce que la réponse soit renvoyée par le service WebSphere MQ.

La figure 40 montre comment l'importation est liée au service externe.



1.

Figure 40. Ressources de liaison d'importation WebSphere MQ

Liaisons d'exportation WebSphere MQ

La liaison d'exportation WebSphere MQ offre les moyens aux modules SCA de fournir des services aux applications externes basées sur WebSphere MQ.

Un bean MDB est déployé pour écouter les demandes parvenant à la **file d'attente de destination de réception** spécifiée dans la liaison d'exportation. La file d'attente spécifiée dans la zone **file d'attente de destination d'envoi** est utilisée pour envoyer la réponse à la demande entrante si le composant appelé fournit une réponse. La file d'attente spécifiée dans la zone replyTo du message de réponse remplace la file d'attente spécifiée dans la zone **file d'attente de destination d'envoi**.

La figure montre comment le demandeur externe est lié à l'exportation.

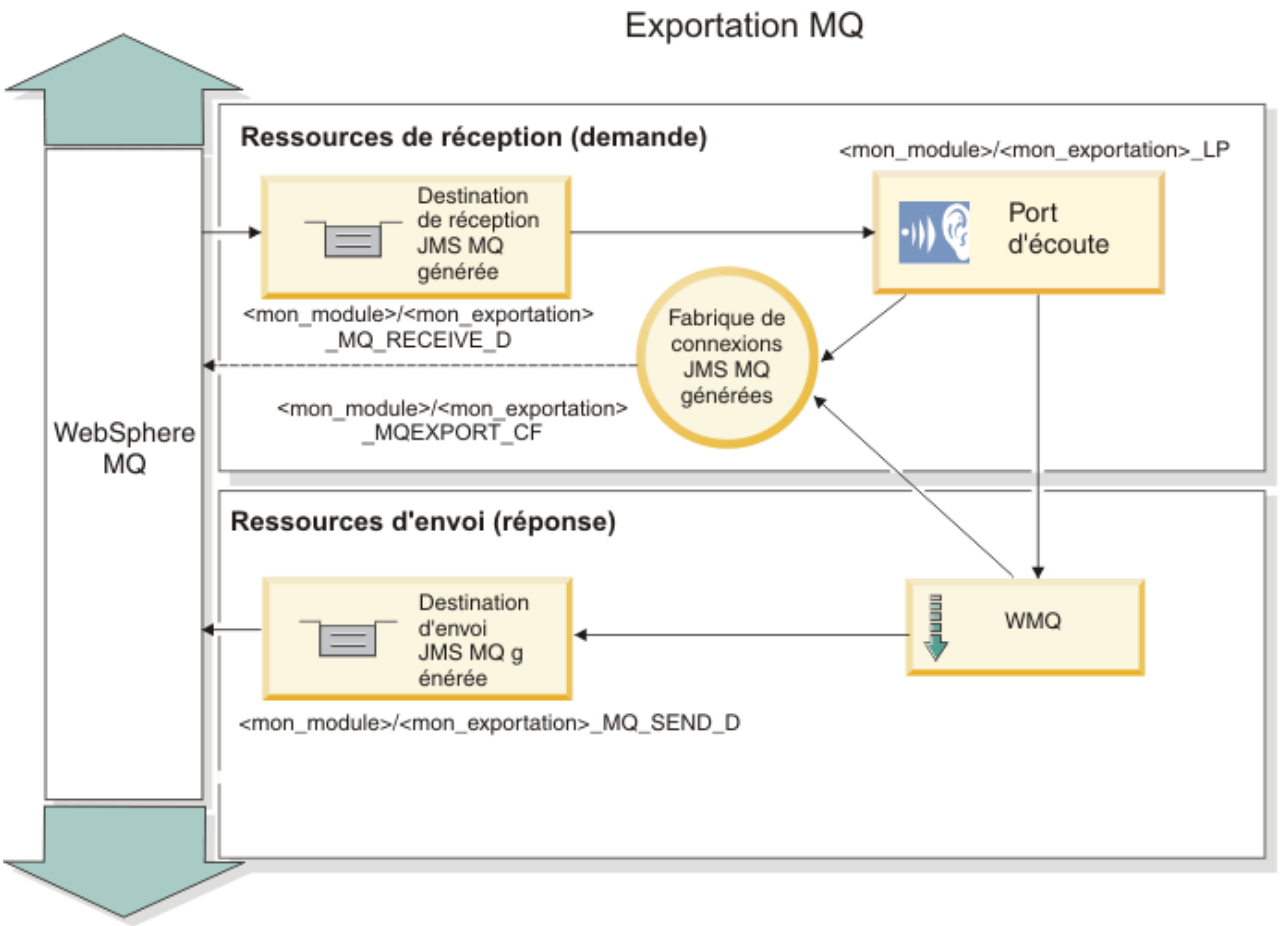


Figure 41. Ressources de liaison d'exportation WebSphere MQ

Remarque : La figure 40, à la page 115 et la figure 41 illustrent comment une application d'une version précédente de WebSphere Process Server est associée à un service externe. Pour les applications développées pour WebSphere Process Server, version 7.0, la spécification d'activation est utilisée à la place du port de l'écouteur et de la fabrique de connexions.

Principales fonctionnalités d'une liaison WebSphere MQ

Les fonctionnalités essentielles d'une liaison WebSphere MQ incluent les en-têtes, les artefacts Java EE et les ressources Java EE créées.

Schémas de corrélation

Une application de demande/réponse WebSphere MQ peut utiliser différentes méthodes de corrélation entre les messages de réponse et les requêtes, qui reposent sur les zone MessageID et CorrelID du descripteur MQMD. Dans la plupart des cas, le demandeur laisse le gestionnaire de file d'attente sélectionner la valeur MessageID, puis l'application qui répond copie cette valeur dans la zone CorrelID de la réponse. En général, le demandeur et l'application qui répond savent implicitement quelle méthode de corrélation est utilisée. Parfois, l'application qui répond satisfait plusieurs indicateurs de la zone Report de la demande qui décrivent la méthode de traitement des zones.

Les liaisons d'exportation des messages WebSphere MQ peuvent être configurées à l'aide des options suivantes :

Options MsgId de la réponse :

New MsgID
Permet à un gestionnaire de file d'attente de sélectionner un MsgId pour la réponse (par défaut).

Copy from Request MsgID
Copie de la zone MsgId depuis la zone MsgId de la demande.

Copy from SCA message
Indique que le MsgId est celui des en-têtes de WebSphere MQ dans le message de réponse SCA, ou bien, en l'absence de valeur, c'est le gestionnaire de file d'attente qui définit un nouvel Id.

As Report Options
La zone Report du descripteur MQMD de la demande est analysée pour déterminer la manière de traiter MsgId. Les options MQRO_NEW_MSG_ID et MQRO_PASS_MSG_ID sont prises en charge et fonctionnent comme Nouveau MsgId et Copie depuis MsgID de demande, respectivement

Options CorrelId de la réponse :

Copy from Request MsgID
Copie de la zone CorrelId depuis la zone MsgId de la demande (par défaut).

Copy from Request CorrelID
Copie de la zone CorrelId depuis la zone CorrelId de la demande.

Copy from SCA message
Indique que CorrelId est celui des en-têtes WebSphere MQ du message de réponse SCA, ou bien, en l'absence de valeur, cette zone reste vide.

As Report Options
La zone Report du descripteur MQMD de la demande est analysée pour déterminer la manière de traiter CorrelId. Les options MQRO_COPY_MSG_ID_TO_CORREL_ID et MQRO_PASS_CORREL_ID sont prises en charge et fonctionnent comme Copy from Request MsgID et Copy from Request CorrelID, respectivement

Les liaisons d'exportation des messages WebSphere MQ peuvent être configurées à l'aide des options suivantes :

Options MsgId de la demande :

New MsgID
Permet à un gestionnaire de file d'attente de sélectionner un MsgId pour la demande (par défaut).

Copy from SCA message
Indique que le MsgId est celui des en-têtes de WebSphere MQ dans le message de requête SCA, ou bien, en l'absence de valeur, permet au gestionnaire de file d'attente de définir un nouvel Id.

Options de corrélation de réponse :

Response has CorrelID copied from MsgId
Dans le message de réponse, la zone CorrelId doit être définie en fonction de la valeur MsgId de la requête (par défaut).

Response has MsgID copied from MsgId
Dans le message de réponse, la zone MsgId doit être définie en fonction de la valeur MsgId de la requête.

Response has CorrelID copied from CorrelId
Dans le message de réponse, la zone CorrelId doit être définie en fonction de la valeur CorrelId de la requête.

Ressources Java EE

Plusieurs ressources Java EE sont créées lorsqu'une liaison WebSphere MQ est déployée dans un environnement Java EE.

Paramètres

Fabrique de connexions MQ
Utilisée par les clients pour créer une connexion au fournisseur WebSphere MQ.

Fabrique de connexions de réponse
Utilisée par l'environnement d'exécution JMS MQ SCA lorsque la destination d'envoi se trouve sur un gestionnaire de files d'attente différent de celui de la destination d'envoi.

Spécification d'activation
Une spécification d'activation MQ JMS est associée à un ou plusieurs beans gérés par message et offre la configuration permettant de recevoir des messages.

Destinations

- Destination d'envoi : Destination à laquelle est envoyé le message de demande ou sortant (importation) ou le message de réponse (exportation), si la valeur n'est pas remplacée par la zone d'en-tête ReplyTo du MQMD du message entrant.
- Destination de réception : Emplacement dans lequel le message de réponse/demande ou le message entrant doit être placé.

En-têtes WebSphere MQ

Les en-têtes WebSphere MQ incorporent certaines conventions pour la conversion aux messages de l'architecture SCA (Service Component Architecture).

Les messages WebSphere MQ se composent d'un en-tête système (le MQMD), de zéro ou plusieurs autres en-têtes MQ (système ou personnalisés) et d'un corps. S'il contient plusieurs en-têtes, leur ordre est important.

Chaque en-tête contient des informations décrivant la structure de l'en-tête suivant.Le MQMD décrit le premier en-tête.

Mode d'analyse des en-têtes MQ

Une liaison de données d'en-tête MQ est utilisée pour analyser les en-têtes MQ. Les en-têtes suivants sont automatiquement pris en charge :

- MQRFH
- MQRFH2
- MQCIH
- MQIIH

Les en-têtes qui commencent par `MQH` sont traités de façon différente. Certains champs de l'en-tête ne sont pas analysés ; ils persistent sous la forme d'octets non analysés.

Pour d'autres en-têtes MQ, vous pouvez écrire des liaisons de données d'en-tête MQ personnalisées afin d'analyser ces en-têtes.

Mode d'accès aux en-têtes MQ

Vous pouvez accéder aux en-têtes MQ de l'une des manières suivantes :

- Via l'objet de message de service (SMO) dans une médiation
- Via l'API ContextService

Les en-têtes MQ sont représentés en interne avec l'élément SMO MQHeader. MQHeader est un conteneur de données d'en-tête qui étend MQControl, mais qui contient un élément de valeur anyType. Il inclut MQMD, MQControl (informations de contrôle du corps de message MQ), ainsi qu'une liste d'autres en-têtes MQ.

- MQMD représente le contenu de la description du message WebSphere MQ, excepté pour les informations déterminant la structure et le codage du corps.
- MQControl contient des informations déterminant la structure et le codage d'un corps de message.
- MQHeaders contient une liste d'objets MQHeader.

La chaîne MQ n'est pas affectée de sorte que, dans le SMO, chaque en-tête MQ inclut ses propres informations de contrôle (CCSID, codage et format). Les en-têtes peuvent être facilement ajoutés ou supprimés, sans modifier d'autres données d'en-tête.

Définition de champs dans le MQMD

Vous pouvez mettre à jour le MQMD à l'aide de l'API Context ou via l'objet de message de service (SMO) dans une médiation. Les champs suivants sont automatiquement propagés vers le message MQ sortant :

- Encoding
- CodedCharacterSet
- Format
- Report
- Expiry
- Feedback
- Priority

- Persistence
- CorrelId
- MsgFlags

Configurez la liaison MQ sur une importation ou une exportation pour propager les propriétés suivantes vers le message MQ sortant :

MsgID
Définissez **Request Message ID** à `copy from SCA message`.

MsgType
Décochez la case **Set message type to MQMT_DATAGRAM or MQMT_REQUEST for request-response operation**.

ReplyToQ
Décochez la case **Override reply to queue of request message**.

ReplyToQMgr
Décochez la case **Override reply to queue of request message**.

A partir de la version 7.0, les champsde contexte peuvent être remplacés à l'aide d'une propriété personnalisée dans la définition de la destination JNDI. Définissez la propriété personnalisée MDCTX avec la valeur `SET_IDENTITY_CONTEXT` dans la destination d'envoi pour propager les champssuivants vers le message MQ sortant :

- UserIdentifier
- AppIdentityData

Définissez la propriété personnalisée MDCTX avec la valeur `SET_ALL_CONTEXT` dans la destination d'envoi pour propager les propriétés suivantes vers le message MQ sortant :

- UserIdentifier
- AppIdentityData
- PutApplType
- PutApplName
- ApplOriginData

Certains champs ne sont pas propagés vers le message MQ sortant. Les champs suivants sont remplacés lors de l'envoi du message :

- BackoutCount
- AccountingToken
- PutDate
- PutTime
- Offset
- OriginalLength

Ajout statique de MQCIH dans une liaison WebSphere MQ

WebSphere Process Server prend en charge l'ajout des informations d'en-tête MQCIH statiquement sans utiliser de module de médiation.

Il existe différentes façons d'ajouter des informations d'en-tête MQCIH à un message (par exemple, en utilisant la primitive de médiation Configurateur d'en-tête). Il peut être utile d'ajouter des informations d'en-tête statiquement, sans utiliser de module de médiation supplémentaire. Les informations d'en-tête

statique, y compris le nom du programme CICS, l'ID de transaction et d'autres détails d'en-tête de format de données peuvent être définis et ajoutés dans le cadre de la liaison WebSphere MQ.

WebSphere MQ, MQ CICS Bridge, et CICS doivent être configurés de telle sorte que les informations d'en-tête MQCIH soient ajoutées statiquement.

Vous pouvez utiliser WebSphere Integration Developer de façon à configurer l'importation WebSphere MQ avec des valeurs statiques qui sont nécessaires pour les informations d'en-tête MQCIH.

Lorsqu'un message arrive et est traité par l'importation WebSphere MQ, une vérification est effectuée pour vérifier que les informations d'en-tête MQCIH figurent déjà dans le message. Si MQCIH est présent, les valeurs statiques définies dans l'importation WebSphere MQ sont utilisées pour remplacer les valeurs dynamiques correspondantes du message. Si MQCIH est absent, Il en est créé un dans le message et les valeurs statiques définies dans l'importation WebSphere MQ sont ajoutées.

Les valeurs statiques définies dans l'importation WebSphere MQ sont spécifiques à une méthode. Vous pouvez spécifier différentes valeurs MQCIH statiques pour différentes méthodes au sein d'une même importation WebSphere MQ.

Cette fonction n'est pas utilisée pour donner des valeurs par défaut si MQCIH ne contient pas les informations d'en-tête spécifiques car une valeur statique définie dans l'importation WebSphere MQ remplacera la valeur correspondante fournie par le message entrant.

Clients externes

WebSphere Process Server peut échanger des messages (envoi et réception) avec des clients externes par le biais de liaisons WebSphere MQ.

Un client externe (comme un portail Web ou un système d'information d'entreprise) peut envoyer un message à un composant SCA dans l'application via une exportation ou il peut être appelé par un composant SCA dans l'application par le biais d'une importation.

La liaison d'exportation WebSphere MQ déploie les beans MDB (Message Driven Beans) pour écouter les demandes entrantes sur la destination receive spécifiée dans la liaison d'exportation. La destination spécifiée dans la zone send est utilisée pour envoyer la réponse à la demande entrante si l'application appelée fournit une réponse. Ainsi, un client externe est en mesure d'appeler des applications par l'intermédiaire d'une liaison d'exportation.

Les importations JMS WebSphere MQ effectuent des liaisons avec des clients externes et peuvent ainsi leur envoyer des messages. Ces messages peuvent ou non exiger une réponse de la part du client externe.

Pour plus d'informations sur l'interaction avec des clients externes à l'aide de WebSphere MQ, reportez-vous au centre d'information WebSphere MQ.

Identification des incidents liés aux liaisons WebSphere MQ

Diagnostic et correction des incidents et des erreurs liés aux liaisons WebSphere MQ.

Conditions d'erreur principales

Les principales conditions d'erreur liées aux liaisons WebSphere MQ peuvent être identifiées via la sémantique transactionnelle, la configuration de WebSphere MQ ou par référence au fonctionnement sur d'autres composants. Les causes premières d'incident peuvent être :

- Erreur de connexion au gestionnaire de file d'attente ou à la file d'attente WebSphere MQ.

  Une erreur de connexion à WebSphere MQ destinée à recevoir des messages entraîne l'échec de démarrage du port d'écoute MDB. Cette situation sera enregistrée dans la journal WebSphere Application Server. Des messages persistants resteront dans la file d'attente de WebSphere MQ jusqu'à ce qu'ils soient récupérés (ou que WebSphere MQ les fasse expirer).

  Une erreur de connexion à WebSphere MQ destinée à envoyer des messages sortants entraîne l'annulation de la transaction qui contrôle l'envoi.
- Erreur d'analyse d'un message entrant ou de construction d'un message sortant.

  Une erreur de liaison des données ou du gestionnaire de données entraîne l'annulation de la transaction qui contrôle le travail.
- Erreur d'envoi du message sortant.

  Une erreur d'envoi d'un message entraîne l'annulation de la transaction en question.
- Messages de réponse multiples ou inattendus.

  L'importation n'attend qu'un seul message de réponse pour chaque message de demande. Si plusieurs réponses sont reçues ou si une réponse tardive (une pour laquelle la réponse SCA a expiré) est reçue, une exception Service Runtime est générée. La transaction est annulée et le message de réponse est retiré de la file d'attente ou traité par le gestionnaire d'événements ayant échoué.

Scénarios d'utilisation incorrecte : comparaison avec les liaisons WebSphere MQ

L'importation et l'exportation WebSphere MQ sont conçues essentiellement pour interopérer avec les applications natives WebSphere MQ et exposer le contenu intégral du corps de message WebSphere MQ aux médiations. Toutefois, la liaison JMS WebSphere MQ est conçue pour l'interopérabilité avec les applications JMS déployées sur WebSphere MQ, où les messages affichés reposent sur un modèle de message JMS.

Dans les scénarios suivants, il est nécessaire d'utiliser la liaison JMSWebSphere MQ et non la liaison WebSphere MQ :

- Appel d'un bean géré par message (MDB) JMS depuis un module SCA, où le MDB est déployé sur le fournisseur JMS WebSphere MQ. Utilisez une importation JMS WebSphere MQ.
- Permettre au module SCA d'être appelé à partir d'un composant de servlet Java EE ou EJB par JMS. Utilisez une exportation JMS WebSphere MQ.
- Médiation du contenu d'un JMS MapMessage, transitant dans WebSphere MQ. Utilisez une exportation et une importation JMS WebSphere MQ conjointement avec la liaison de données appropriée.

Dans certaines situations, la liaison WebSphere MQ et la liaison JMSWebSphere MQ peuvent interopérer. En particulier, si vous reliez des applications WebSphere MQ Java EE et non Java EE, utilisez une exportation WebSphere MQ et une importation JMS WebSphere MQ (ou vice-versa) conjointement avec les liaisons de

données et/ou les modules de médiation appropriés (ou les deux).

Messages non délivrés

Si WebSphere MQ ne parvient pas envoyer un message à la destination prévue (en règle générale, suite à des erreurs de configuration), il envoie le message à une file d'attente de rebut.

Dans ce cas, il ajoute un en-tête de non-distribution au début du corps de message. Ce dernier indique les raisons de l'erreur, la destination d'origine, ainsi que d'autres informations.

Messages SCA basés sur MQ qui n'apparaissent pas dans le gestionnaire des événements ayant échoué

Si des messages SCA ont été générés en raison d'un incident d'interaction WebSphere MQ, vous devriez les retrouver dans le gestionnaire des événements ayant échoué. Si ces messages ne s'affichent pas dans le gestionnaire des événements ayant échoué, vérifiez que la destination WebSphere MQ sous-jacente possède une valeur du nombre maximal de livraisons ayant échoué supérieure à 1. Spécifier une valeur supérieure ou égale à 2 permet une interaction avec le gestionnaire d'événements ayant échoué au cours des appels SCA pour les liaisons WebSphere MQ.

Les événements MQ ayant échoué sont lus à nouveau sur le gestionnaire de files d'attente incorrect.

Quand une fabrique de connexions prédéfinie est utilisée pour les connexions sortantes, les propriétés de connexion doivent correspondre à celles définies dans la spécification d'activation employée pour les connexions entrantes.

La fabrique de connexions prédéfinies est utilisée pour créer une connexion quand un événement ayant échoué est relu. Elle doit donc être configurée pour utiliser le même gestionnaire de files d'attente duquel le message a été reçu à l'origine.

Gestion des exceptions

La configuration de la liaison détermine la manière dont sont gérées les exceptions qui sont émises par les gestionnaires de données ou les liaisons de données. En outre, la nature du flux de médiation détermine le comportement du système lorsqu'une exception de ce type est générée.

Différents incidents peuvent se produire lorsqu'un gestionnaire de données ou une liaison de données est appelé(e) par votre liaison. Par exemple, un gestionnaire de données peut recevoir un message dont la charge est endommagée ou il peut essayer de lire un message dont le format n'est pas correct.

Le traitement d'une telle exception par votre liaison est déterminé par la manière dont vous implémentez le gestionnaire de données ou la liaison de données. Il est ainsi recommandé de configurer votre liaison de données afin qu'elle génère une exception de type `DataBindingException`.

La situation est similaire pour un gestionnaire de données. Dès lors que le gestionnaire de données est appelé par la liaison de données, toute exception de gestionnaire de données est encapsulée dans une exception de liaison de données. Ainsi, une exception `DataHandlerException` vous est signalée sous la forme d'une exception `DataBindingException`.

Lorsqu'une exception d'exécution, y compris une exception `DataBindingException` est émise :

- Si le flux de médiation est configuré pour être transactionnel, le message JMS est stocké dans le gestionnaire des événements ayant échoué par défaut pour une relecture manuelle ou une suppression.

  Remarque : Vous pouvez modifier le mode de récupération sur la liaison afin que le message soit annulé plutôt que stocké dans le gestionnaire des événements ayant échoué.
- Si le flux de médiation n'est pas de type transactionnel, l'exception est consignée et le message est perdu.

La situation est similaire pour un gestionnaire de données. Le gestionnaire de données étant appelé par la liaison de données, une exception de gestionnaire de données est générée dans une exception de liaison de données. Par conséquent, une exception `DataHandlerException` vous est signalée sous la forme d'une exception `DataBindingException`.

Limitations des liaisons

L'utilisation des liaisons connaît certaines limitations qui sont répertoriées ici.

Concepts associés

«Limitations de la liaison MQ»
L'utilisation de la liaison MQ connaît certaines limitations qui sont répertoriées ici.

«Limitations des liaisons JMS, JMS MQ et JMS génériques», à la page 125
Les liaisons JMS et JMS MQ connaissent certaines limitations.

Limitations de la liaison MQ

L'utilisation de la liaison MQ connaît certaines limitations qui sont répertoriées ici.

Pas de distribution de messages par publication/abonnement

La méthode de distribution des messages par publication/abonnement n'est pas prise en charge par la liaison MQ, bien que WMQ la prenne en charge. Toutefois, la liaison JMS MQ prend en charge cette méthode de distribution.

Files d'attentes de réception partagées

Plusieurs liaisons d'exportation et d'importation WebSphere MQ attendent que tous les messages présents dans la file d'attente de réception configurée soient destinés à l'exportation ou l'importation. Les liaisons d'importation et d'exportation doivent être configurées en prenant en compte ce qui suit :

- Chaque importation MQ doit avoir une file d'attente de réception distincte, car les liaisons d'importation MQ supposent que tous les messages dans la file d'attente de réception sont des réponses à des demandes qu'elles ont envoyées. Si la file d'attente de réception est partagée par plusieurs importations, les réponses peuvent être reçues par la mauvaise importation et ne seront pas mises en corrélation avec le message de demande d'origine.
- Chaque exportation MQ doit avoir une file d'attente de réception distincte, sinon vous ne pouvez pas prévoir quelle exportation recevra un message de demande déterminé.
- Les importations et les exportations MQ peuvent désigner la même file d'attente d'envoi.

Limitations des liaisons JMS, JMS MQ et JMS génériques

Les liaisons JMS et JMS MQ connaissent certaines limitations.

Implications de la génération de liaisons par défaut

Les limitations de l'utilisation de liaisons JMS, JMS MQ et JMS génériques sont présentées dans les sections suivantes :

- Implications de la génération de liaisons par défaut
- Schéma de corrélation des réponses
- Support bidirectionnel

Lorsque vous générez une liaison, plusieurs zones sont renseignées automatiquement par défaut, si vous ne choisissez pas d'entrer les valeurs vous-même. Par exemple, un nom de fabrique de connexions est créé automatiquement. Si vous savez que vous placerez votre application sur un serveur et que vous y accéderez à distance à l'aide d'un client, vous devez, lors de la création de la liaison, entrez des noms JNDI plutôt que d'utiliser les valeurs par défaut car vous souhaiterez certainement contrôler ces valeurs via la console d'administration lors de la phase d'exécution.

Toutefois, si vous avez accepté les valeurs par défaut, mais que vous découvrez par la suite que vous ne pouvez pas accéder à votre application à partir d'un client éloigné, vous pouvez utiliser la console d'administration pour définir explicitement la valeur de la fabrique de connexions. Recherchez la zone des noeuds finaux du fournisseur dans les paramètres de la fabrique de connexions et ajoutez une valeur telle que la suivante : <nomhôte_serveur>:7276 (si vous utilisez le numéro de port par défaut).

Schéma de corrélation des réponses

Si vous utilisez le schéma de corrélation des réponses CorrelationId à CorrelationId, permettant de corréler les messages dans une opération demande/réponse, le message doit contenir un ID corrélation dynamique.

Pour créer un ID corrélation dynamique dans un module de médiation à l'aide de l'éditeur de flux de médiation, ajoutez un noeud XSLT avant l'importation avec la liaison JMS. Ouvrez l'éditeur de mappage XSLT. Les en-têtes connus de l'architecture des composants de service seront disponibles dans le message cible. Faites glisser une zone contenant un ID unique dans le message dans l'ID corrélation de l'en-tête JMS du message cible.

Support bidirectionnel

Seuls les caractères ASCII sont pris en charge pour les noms JNDI (Java Naming and Directory Interface) lors de la phase d'exécution.

Files d'attentes de réception partagées

Plusieurs liaisons d'exportation et d'importation attendent que tous les messages présents dans la file d'attente de réception configurée soient destinés à l'exportation ou l'importation. Les liaisons d'importation et d'exportation doivent être configurées en prenant en compte ce qui suit :

- Chaque liaison d'importation doit avoir une file d'attente de réception distincte, car ces liaisons supposent que tous les messages dans la file d'attente de réception sont des réponses à des demandes qu'elles ont envoyées. Si la file

d'attente de réception est partagée par plusieurs importations, les réponses peuvent être reçues par la mauvaise importation et ne seront pas mises en corrélation avec le message de demande d'origine.

- Chaque exportation doit avoir une file d'attente de réception distincte, sinon vous ne pouvez pas prévoir quelle exportation recevra un message de demande déterminé.
- Les importations et les exportations peuvent désigner la même file d'attente d'envoi.

Chapitre 3. Guides et techniques de programmation

Cette section comprend des guides et des exemples de programmation.

Les sous-rubriques ci-après fournissent des informations pour la programmation de divers composants, applications et solutions d'intégration métier.

Important : Voir la section Référence du centre de documentation pour obtenir des détails sur les API (interfaces de programme d'application) et les SPI a(interfaces de programmation de système) qui sont prises en charge par WebSphere Process Server et WebSphere Enterprise Service Bus.

Concepts associés

«Programmation SCA (Service Component Architecture)»
SCA (Service Component Architecture) offre un modèle de programmation simple, mais puissant, permettant de créer des applications basées sur l'architecture orientée services (SOA).

«Programmation des objets métier», à la page 174
Les objets métier sont des conteneurs de données d'application, telles qu'un client ou une facture. Les données sont échangées entre les composants par l'intermédiaire d'objets métier. La structure sous-jacente d'un objet métier est une définition de schéma XML (XSD) et l'accès par programmation aux objets métier est fourni via des interfaces d'objet métier dans WebSphere. Collectivement, ces aspects de l'objet métier, sa représentation structurelle, ses interfaces de programmation, ainsi que son comportement et sa manipulation dans SCA (Service Component Architecture), représentent la structure de l'objet métier, qui fournit des moyens puissants et cohérents de décrire et distribuer les données métier dans votre solution.

«Programmation de la gestion des règles métier», à la page 233
Des classes de gestion des règles métier sont fournies pour permettre de créer des clients de gestion personnalisés ou d'automatiser les changements apportés aux règles métier.

«Programmation de widgets», à la page 349
Business Space fournit des widgets qui sont activés pour divers produits IBM de gestion des processus métier. Toutefois, vous pouvez également créer vos propres widgets et les intégrer avec ceux d'IBM. Les références ci-après expliquent pourquoi créer des widgets et comment les créer, et fournit des références aux API que vous pouvez utiliser pour créer vos propres widgets.

Programmation SCA (Service Component Architecture)

SCA (Service Component Architecture) offre un modèle de programmation simple, mais puissant, permettant de créer des applications basées sur l'architecture orientée services (SOA).

Concepts associés

«SCDL (Service Component Definition Language)»
SCDL (Service Component Definition Language) est un langage XML permettant de décrire des éléments SCA (Service Component Architecture), tels que des modules, des composants, des références, des importations et des exportations.

«Principes de base du modèle de programmation SCA», à la page 139
Le concept d'un *composant* logiciel est à la base du modèle de programmation SCA (Service Component Architecture). Un composant est une unité qui implémente une logique et qui la rend disponible aux autres composants via une interface. Un composant peut également exiger les services rendus disponibles par les autres composants. Dans ce cas, le composant affiche une *référence* pour ces services.

«Techniques de programmation SCA», à la page 170
Cette section fournit des exemples de technique de programmation SCA (Service Component Architecture).

SCDL (Service Component Definition Language)

SCDL (Service Component Definition Language) est un langage XML permettant de décrire des éléments SCA (Service Component Architecture), tels que des modules, des composants, des références, des importations et des exportations.

Les divers types d'artefact qui existent dans SCA ont été conçus pour prendre en charge certaines des exigences de base de l'architecture orientée services. Pour commencer, SCA a besoin d'un mécanisme pour définir un composant de service de base. Une fois qu'il existe un mécanisme de définition des composants de service, il est important de pouvoir rendre ces services accessibles aux clients qui se trouvent à l'intérieur ou à l'extérieur du module SCA actuel. De plus, il doit exister une construction pour importer et référencer les services externes au module SCA actuel. Enfin, SCA offre des constructions pour regrouper des services et modules dans des applications plus importantes.

Les définitions SCDL sont organisées sur plusieurs fichiers. Par exemple, dans une application d'approbation de crédit, nous pouvons stocker le SCDL de l'interface et l'implémentation dans un fichier appelé `CreditApproval.component`. Les références peuvent être incluses dans le fichier `CreditApproval.component` (en ligne) ou dans un fichier `sca.references` distinct, à la racine du module. Toute référence autonome est placée dans le fichier `sca.references`. Les références autonomes peuvent être utilisées par des artefacts non SCA (JSP) dans le même SCA pour appeler le composant SCA.

Tableau 14. Artefacts principaux constituant un module de service SCA

Artefact	Définition SCDL
Définition de module	• Contenue dans le fichier `sca.module` à la racine du fichier JAR du projet SCA
Composants de service	• Un module peut contenir 0..n définitions de service • Chaque définition de composant est contenue dans un fichier `<NOM_SERVICE>.component`
Importations	• Un module peut contenir 0..n définitions d'importation • Chaque définition d'importation est contenue dans un fichier `<NOM_IMPORTATION>.import`

Tableau 14. Artefacts principaux constituant un module de service SCA (suite)

Artefact	Définition SCDL
Exportations	• Un module peut contenir 0..n définitions d'exportation • Chaque définition d'exportation est contenue dans un fichier `<NOM_EXPORTATION>.export`
Références	• Deux types de référence – En ligne (dans une définition de composant de service) – Autonome • Chaque définition de composant est contenue dans un fichier `<NOM_SERVICE>.reference`
Autres artefacts	• Les autres artefacts incluent : les classes Java, les fichiers WSDL, les autres fichiers XSD d'artefact, BPEL.

Lors de la génération d'une application SCA, WebSphere Integration Developer génère les définitions SCDL appropriées. Toutefois une connaissance de base du langage SCDL peut vous aider à comprendre l'architecture globale et à participer au débogage des applications.

Concepts associés

«Définition de module»
SCA (Service Component Architecture) définit un modèle de déploiement standard pour l'intégration de composants dans un module de service. Le fichier `sca.module` contient la définition du module.

«Définition de composant», à la page 131
La définition de composant de service est incluse dans un fichier appelé `<NOM_SERVICE>.component`. Les composants SCA et leurs dépendances peuvent être définies et intégrées ensemble dans des unités déployables.

«Définition d'importation», à la page 134
La définition d'importation est incluse dans un fichier appelé `<NOM_IMPORTATION>.import`. Les importations SCA permettent aux clients d'un module SCA d'accéder à des services externes au module SCA actuel.

«Définition d'exportation», à la page 136
La définition d'exportation est incluse dans un fichier appelé `<NOM_EXPORTATION>.export`. Les exportations SCA permettent d'accéder aux composants de service définis dans un module SCA pour qu'ils puissent être utilisés par des clients externes au module SCA actuel.

«Définition de référence», à la page 137
Les clients SCA et non SCA qui appellent un composant de service ont besoin d'une référence à ce service pour l'appeler. Les références peuvent être définies comme références autonomes dans le fichier `sca.reference` ou comme références en ligne dans une définition de composition de service.

Définition de module

SCA (Service Component Architecture) définit un modèle de déploiement standard pour l'intégration de composants dans un module de service. Le fichier `sca.module` contient la définition du module.

Un module SCA n'est pas seulement un autre type de package. Dans WebSphere Process Server, un module de service SCA est équivalent à un fichier EAR Java EE et plusieurs autres sous-modules Java EE. Les éléments Java EE, tels qu'un fichier WAR, peuvent être intégrés au module SCA. Des artefacts non SCA (pages JSP et autres) peuvent également être intégrés à un module de service SCA, ce qui leur permet d'appeler des services SCA à l'aide du modèle de programmation client SCA grâce à un type de référence spéciale appelée référence autonome.

Voici un exemple de fichier `sca.module` :

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:module xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
name="CreditApproval"/>
```

Le diagramme illustre un module de WebSphere Integration Developer avec sa définition de module SCDL associée que vous pouvez afficher dans un éditeur. Dans cet exemple, le module correspond à un module de médiation.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:module xmlns:mt="http://www.ibm.com/xmlns/prod/websphere/scdl/moduletype/7.0.0"
  Xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"name="StockQuote">
  <mt:moduleType type="com.ibm.ws.sca.scdl.moduletype.mediation" version="1.0.0"/>
</Scdl:module>
```

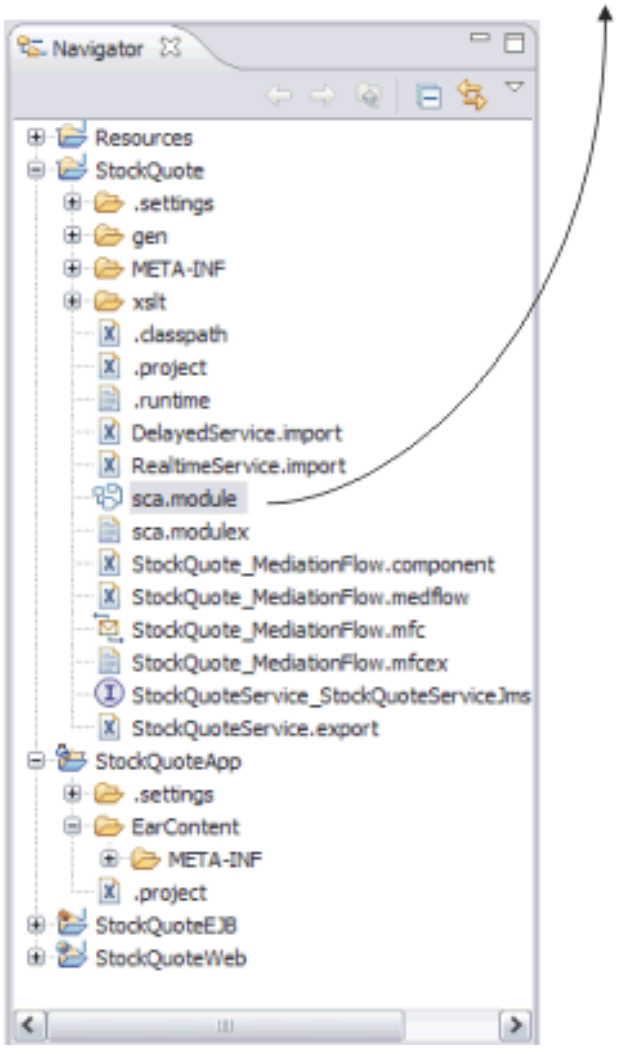


Figure 42. Illustration de la relation entre un module de WebSphere Integration Developer et la définition de module interne

Définition de composant

La définition de composant de service est incluse dans un fichier appelé `<NOM_SERVICE>.component`. Les composants SCA et leurs dépendances peuvent être définies et intégrées ensemble dans des unités déployables.

Cette figure décrit de manière plus détaillée la définition de composant de service. Chaque composant de service doit posséder un nom unique dans le module SCA et doit correspondre au chemin d'accès au fichier relatif à la racine du module. Comme indiqué dans la diapositive précédente, la définition de composant de service est incluse dans un fichier appelé `<NOM_SERVICE>.component`.

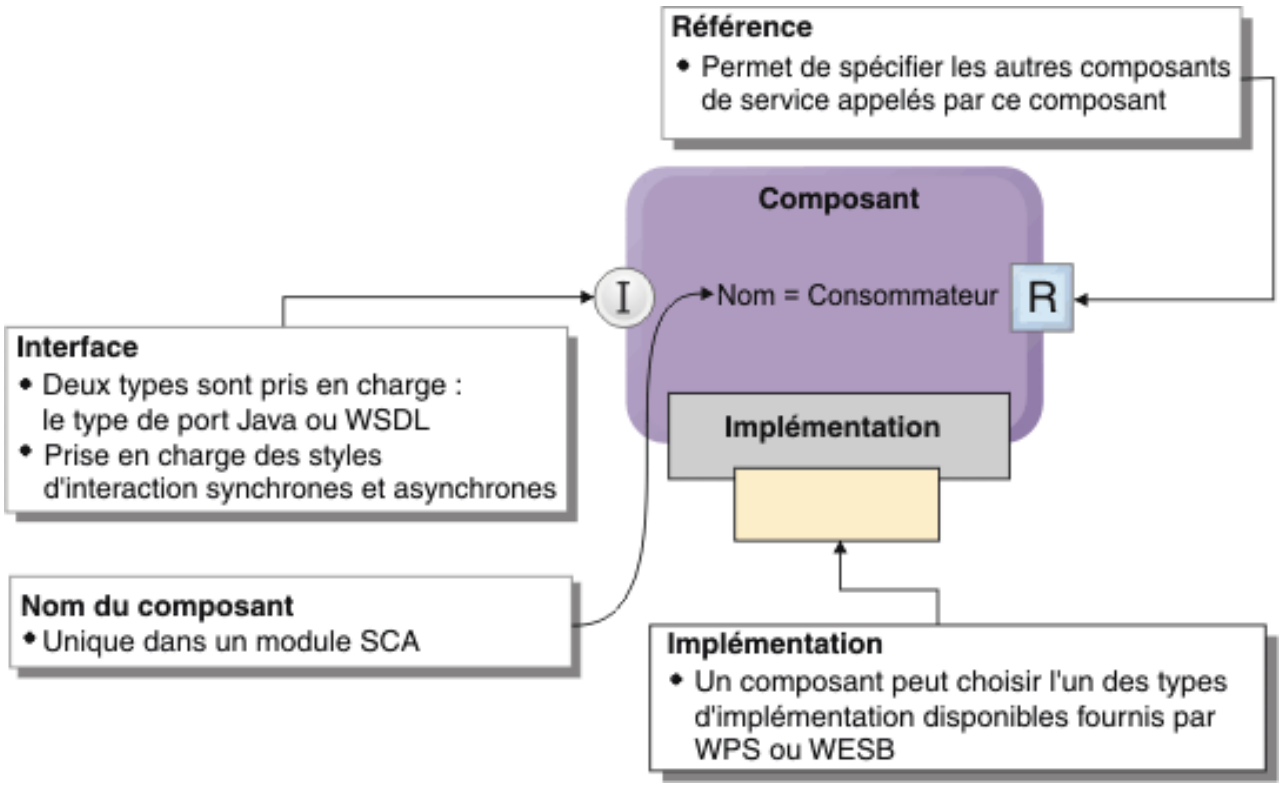
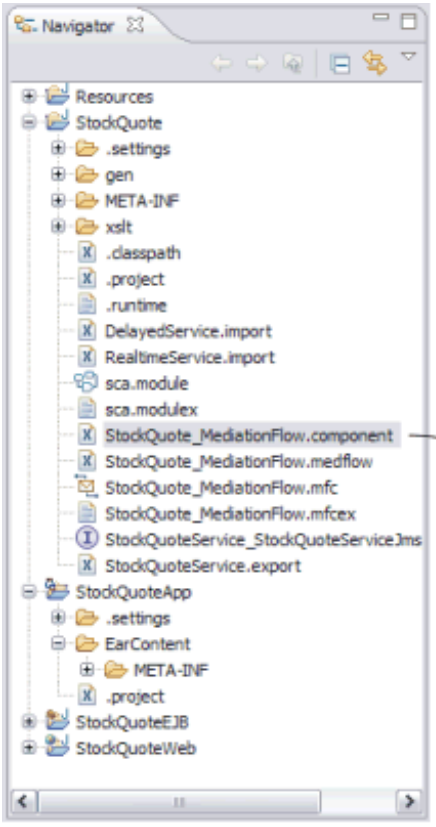


Figure 43. Définition de composant de service avec le nom du composant, l'implémentation, les interfaces et les références

Chaque composant de service doit posséder un nom unique dans le module SCA et doit correspondre au chemin d'accès au fichier relatif à la racine du module. Ensuite, chaque composant de service peut être associé à une ou plusieurs interfaces, qui peuvent correspondre à des définitions d'interface de type Java ou Port WSDL. Les interfaces associées à un composant de service peuvent prendre en charge les interactions de style synchrone ou asynchrone avec les clients qui appellent le service.

Chaque composant de service peut être implémenté de diverses manières, spécifiées par la définition d'implémentation. Enfin, les composants de service peuvent appeler d'autres composants de service ou importations définis dans le module de service actuel. Dans ce cas, la référence appropriée doit être définie pour indiquer quel service est utilisé. Souvent, ce type de référence est en ligne dans la définition de composant de service, bien qu'elle puisse également être placée dans le fichier des références autonomes. Chaque composant de service peut posséder aucune, une ou plusieurs références à d'autre services appelés par le composant de service défini.

Voici un exemple qui illustre la définition du composant StockQuote_MediationFlow. Notez que le composant inclut les définitions d'une interface WSDL, de deux références et d'une implémentation.




```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        xmlns:ns1="http://Resources/StockQuoteService"
        xmlns:ns2="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:ns3="http://stockquote.samp.sibx.websphere.ibm.com/RealtimeService/"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        DisplayName="StockQuote_MediationFlow" name="StockQuote_MediationFlow">
 <interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:StockQuoteService"/>
 </interfaces>
 <references>
  <reference name="DelayedServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:DelayedServicePortType">
    <method name="getQuote"/>
   </interface>
   <wire target="DelayedService"/>
  </reference>
  <reference name="RealtimeServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns3:RealtimeServicePortType">
    <Method name="getQuote"/>
   </interface>
   <wire target="RealtimeService"/>
  </reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="StockQuote_MediationFlow.mfc"/>
</Scdl:component>
```

Figure 44. Exemple de définition de composant de service dans Service Component Definition Language

Définition d'importation

La définition d'importation est incluse dans un fichier appelé `<NOM_IMPORTATION>.import`. Les importations SCA permettent aux clients d'un module SCA d'accéder à des services externes au module SCA actuel.

Comme les composants de service, les importations possèdent un nom et un ensemble d'interfaces 1..N auxquelles elles sont associées. Les importations possèdent également un attribut de liaison, qui permet de décrire comment le service externe est lié au module actuel. Les types de liaison courants sont indiqués dans le diagramme.

Les importations correspondent en fait à un type de composant de service spécial dans un module SCA. Les importations sont des cibles valides dans une définition de connexion d'une référence de service. Cela signifie que pour un client qui appelle un service cible, le modèle de programmation du client est le même, que la référence pointe vers une importation ou un autre composant de service.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:ns1="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        xmlns:webservice="http://www.ibm.com/xmlns/prod/websphere/scdl/webservice/7.0.0"
        xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        displayName="DelayedService" name="DelayedService">
 <interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:DelayedServicePortType">
   <method name="getQuote"/>
  </interface>
 </interfaces>
 <esbBinding xsi:type="webservice:WebServiceImportBinding"
        endpoint="http://localhost:9080/DelayedService/services/DelayedServiceSOAP"
        port="ns1:DelayedServiceSOAP" service="ns1:DelayedService"/>
</scdl:import>
```

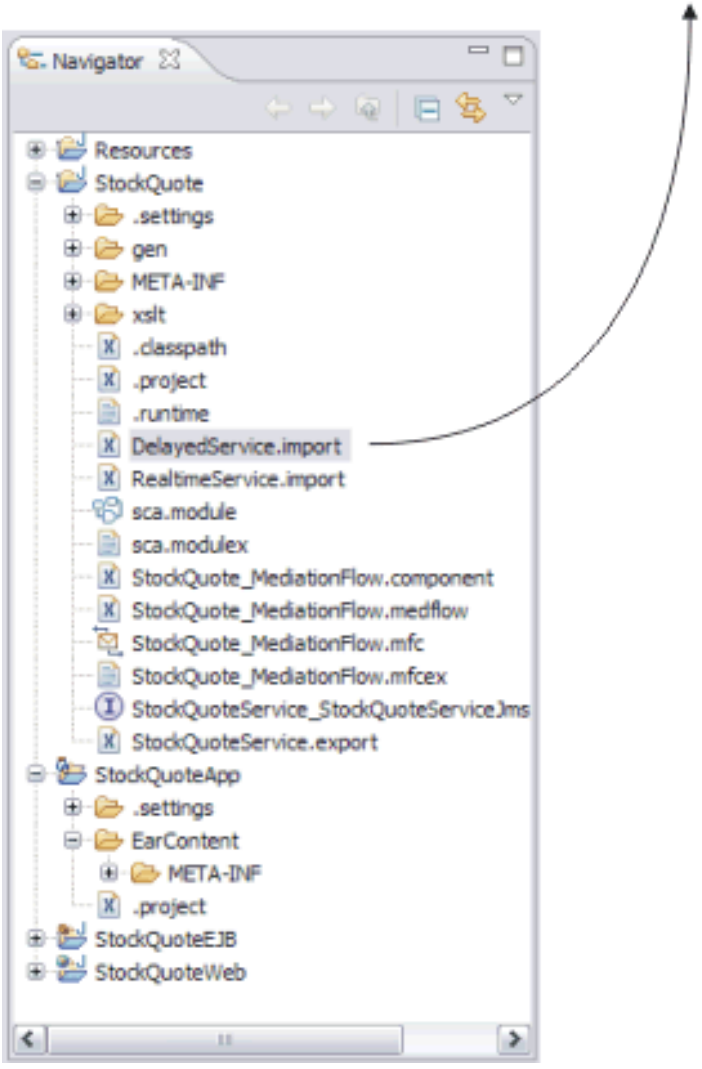


Figure 45. Exemple de définition d'importation dans Service Component Definition Language

Définition d'exportation

La définition d'exportation est incluse dans un fichier appelé `<NOM_EXPORTATION>.export`. Les exportations SCA permettent d'accéder aux composants de service définis dans un module SCA pour qu'ils puissent être utilisés par des clients externes au module SCA actuel.

Les composants d'exportation incluent un nom et un attribut cible, qui désigne le composant de service à exporter. Comme les composants d'importation, les exportations possèdent un attribut de liaison qui indique comment le service est lié en externe. Les types de liaison courants sont indiqués dans le diagramme.

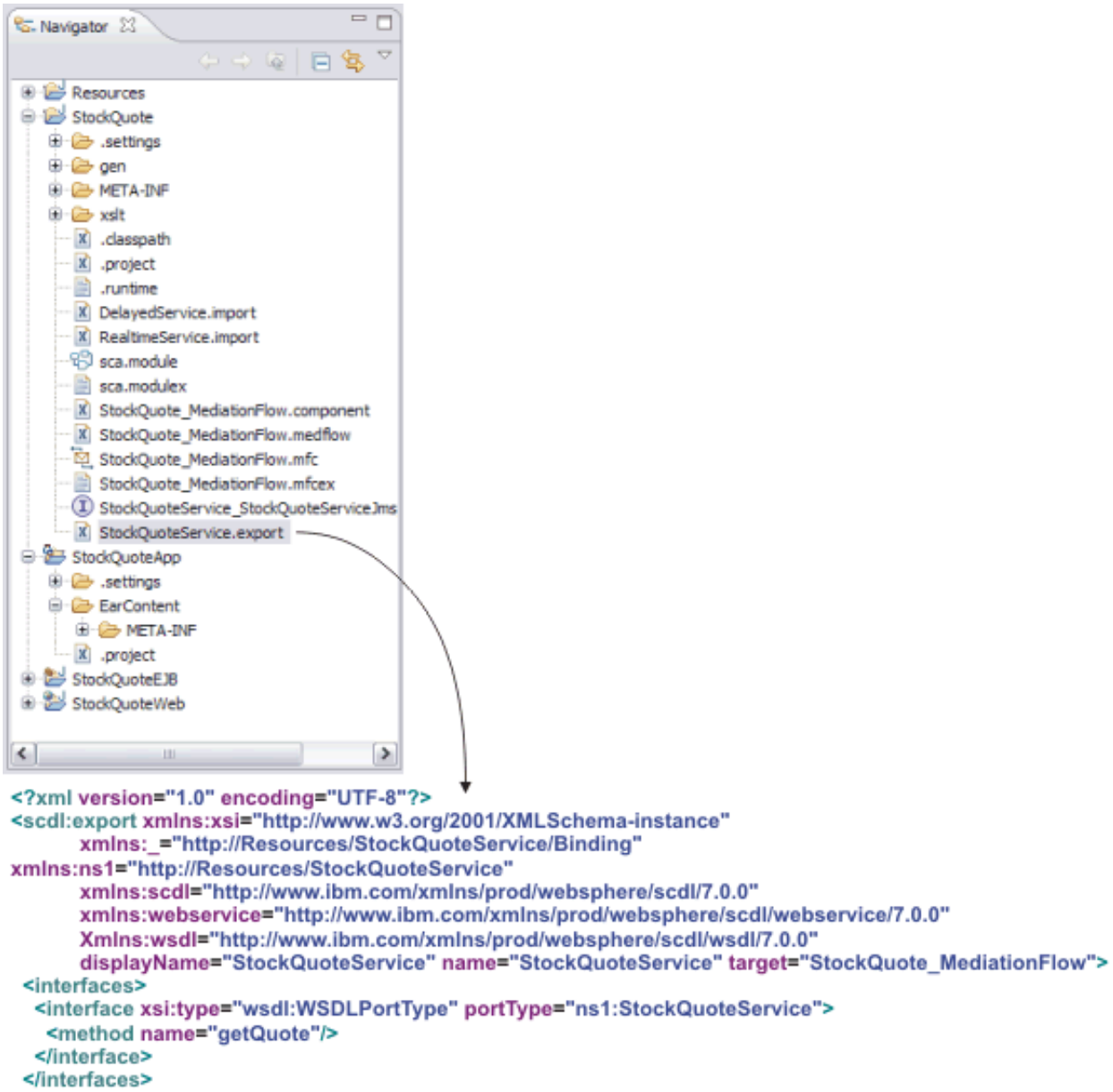


Figure 46. Exemple de définition d'exportation dans Service Component Definition Language

Définition de référence

Les clients SCA et non SCA qui appellent un composant de service ont besoin d'une référence à ce service pour l'appeler. Les références peuvent être définies comme références autonomes dans le fichier `sca.reference` ou comme références en ligne dans une définition de composition de service.

Chaque référence possède un nom, qui permet à un client de rechercher le service approprié à l'aide du modèle de programmation du client. En plus du nom, une référence inclut également un élément d'interface. La multiplicité d'une référence indique le nombre de définitions de connexion qui peuvent désigner cette référence comme source. Enfin, la définition de connexion spécifie le nom du composant de service cible ou de l'importation qui résout la référence.

Il existe deux manières de définir des références. La première consiste à placer la référence en ligne dans la définition de composant de service. Si cette approche est utilisée, les références ne sont disponibles que pour le composant de service dans lequel les références sont incluses. Une autre approche consiste à inclure les définitions de référence dans le fichier des références autonomes. Pour cette approche, les références peuvent être utilisées par un client non SCA ou un autre composant du module. Un composant d'interface graphique tel qu'une page JSP qui a besoin de pouvoir appeler un service particulier représente un exemple de composant non SCA qui utilise une référence du fichier des références autonomes. Pour appeler un composant de service, le client a besoin d'une référence afin de pouvoir utiliser l'environnement d'exécution SCA pour rechercher le service approprié à appeler.

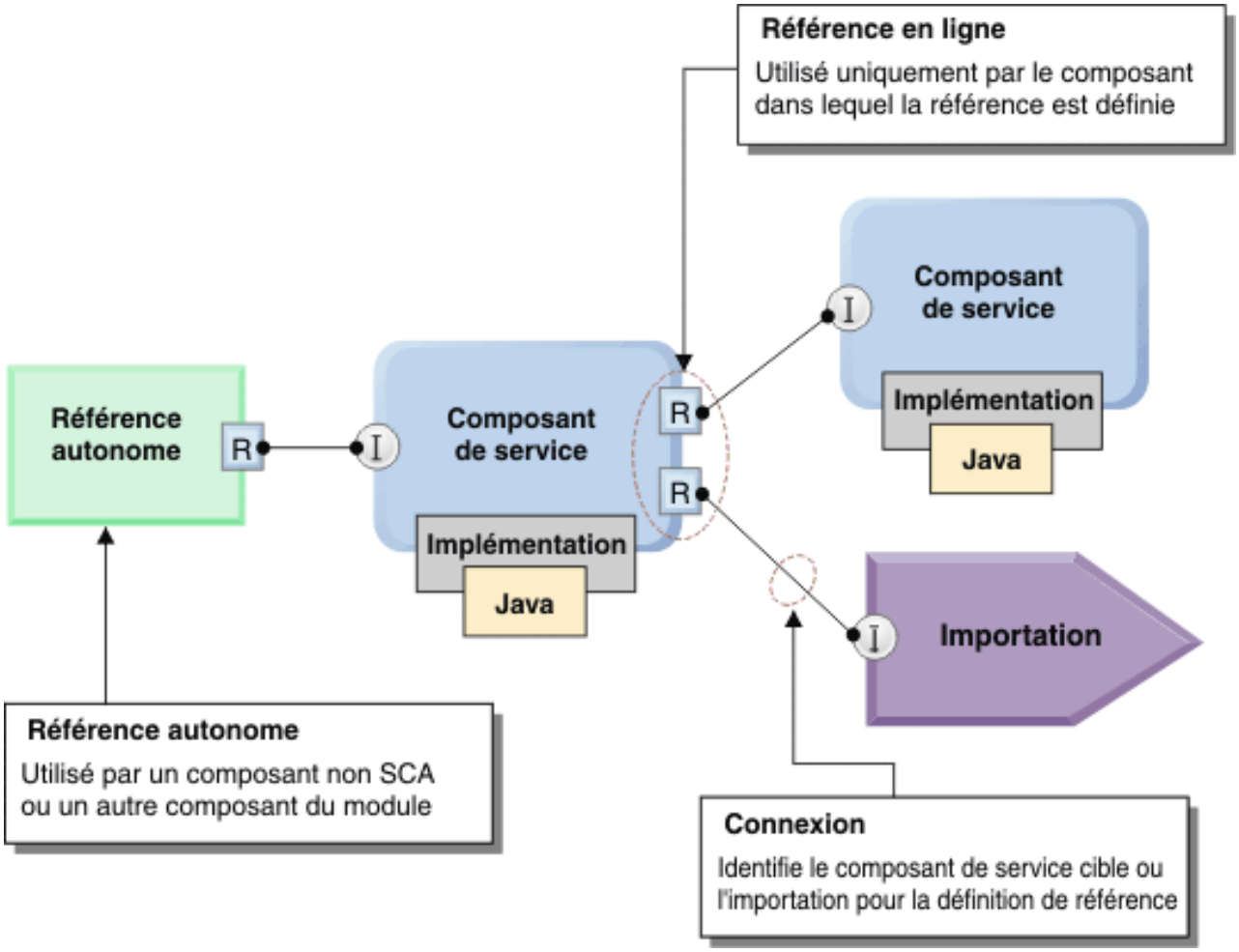


Figure 47. Les clients peuvent appeler un composant de service à l'aide de références autonomes ou en ligne

Voici un exemple qui illustre la définition du composant qui inclut deux références en ligne : DelayedServicePortTypePartner et RealtimeServicePortTypePartner. Notez que le composant inclut les définitions d'une interface WSDL, de deux références et d'une implémentation.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        Xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        Xmlns:ns1="http://Resources/StockQuoteService"
        xmlns:ns2="http://stockquote.samp.sibx.websphere.ibm.com/DelayedService/"
        xmlns:ns3="http://stockquote.samp.sibx.websphere.ibm.com/RealtimeService/"
        Xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        DisplayName="StockQuote_MediationFlow" name="StockQuote_MediationFlow">
 <Interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:StockQuoteService"/>
 </Interfaces>
 <references>
  <reference name="DelayedServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:DelayedServicePortType">
    <method name="getQuote"/>
   </interface>
   <wire target="DelayedService"/>
  </reference>
  <reference name="RealtimeServicePortTypePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns3:RealtimeServicePortType">
    <Method name="getQuote"/>
   </interface>
   <wire target="RealtimeService"/>
  </reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="StockQuote_MediationFlow.mfc"/>
</Scdl:component>
```

Figure 48. Exemple de définitions de référence en ligne

Principes de base du modèle de programmation SCA

Le concept d'un *composant* logiciel est à la base du modèle de programmation SCA (Service Component Architecture). Un composant est une unité qui implémente une logique et qui la rend disponible aux autres composants via une interface. Un composant peut également exiger les services rendus disponibles par les autres composants. Dans ce cas, le composant affiche une *référence* pour ces services.

Dans SCA, chaque composant doit afficher au moins une interface. Le diagramme d'assemblage de la figure 49, à la page 140 dispose de trois composants. Chacun d'entre eux dispose d'une interface représentée par la lettre I entourée d'un cercle. Un composant peut également se référer à d'autres composants. Les références sont représentées par la lettre R entourée d'un carré. Les références et les interfaces sont ensuite liées dans un diagramme d'assemblage. En règle générale, le développeur d'intégration "résout" les références en les connectant aux interfaces de composants qui implémentent la logique requise.

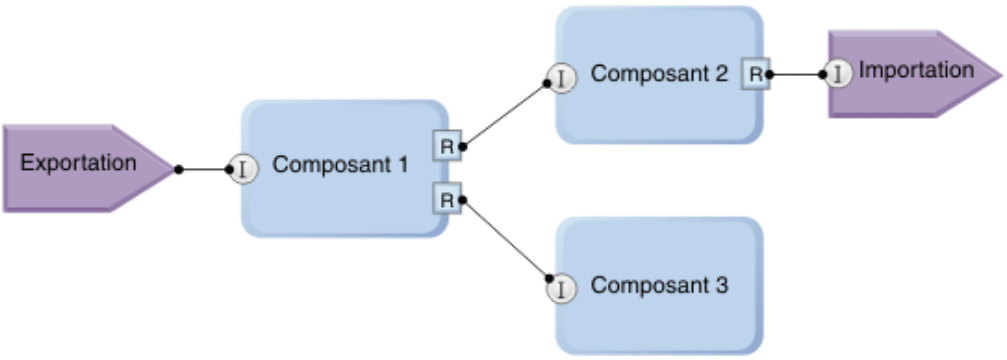


Figure 49. Diagramme d'assemblage

Appel de composants SCA

Pour fournir un accès aux services à appeler, le modèle de programmation SCA contient une classe *ServiceManager* qui permet aux développeurs de rechercher les services disponibles par nom. Voici un fragment de code Java type illustrant la recherche d'un service. La classe ServiceManager permet d'obtenir une référence au service BOFactory qui est un service fourni par le système :

```
//Get service manager singleton
ServiceManager smgr = new ServiceManager();
//Access BOFactory service
BOFactory bof =(BOFactory)
        smgr.locateService("com/ibm/websphere/bo/BOFactory");
```

Remarque : Le package de la classe ServiceManager est `com.ibm.websphere.sca`.

Les développeurs peuvent utiliser un mécanisme similaire pour obtenir les références de leurs propres services en spécifiant le nom du service référence dans la méthode *locateService*. Après avoir obtenu la référence d'un service à l'aide de la classe ServiceManager, vous pouvez invoquer n'importe quelle opération disponible sur ce service indépendamment du protocole d'appel et du type d'implémentation.

Pour appeler des composants SCA, il existe trois styles d'appel différents :

- **Appel synchrone** : Lorsque vous utilisez ce style d'appel, l'appelant attend de façon synchrone que la réponse soit envoyée. Il s'agit du mécanisme d'appel classique.
- **Appel asynchrone** : Ce mécanisme permet à l'appelant d'appeler un service sans attendre l'envoi immédiat d'une réponse. Au lieu d'obtenir une réponse, l'appelant obtient un "ticket" qui peut être utilisé ultérieurement pour récupérer la réponse. L'appelant récupère la réponse en appelant une opération spéciale qui doit être fournie par l'appelé dans ce cas de figure.
- **Appel asynchrone avec rappel** : Ce style d'appel est similaire au précédent, mais l'appelé est chargé de l'envoi de la réponse. L'appelant doit afficher une opération spéciale (opération de rappel) que l'appelé peut appeler lorsque la réponse est prête.
- **Appel asynchrone avec réponse différée** : Dans ce style d'appel, le client appelle un service, puis continue le traitement jusqu'à ce que le client envoie une demande pour capturer la réponse.

Importations

Parfois, les composants ou les fonctions disponibles sur des systèmes externes indiquent la logique métier, comme les applications existantes ou d'autres implémentations externes. Dans ce cas, le développeur d'intégration ne peut pas résoudre la référence en connectant une référence à un composant qui contient l'implémentation dont il/elle a besoin pour connecter la référence à un composant qui "pointe vers" l'implémentation externe. Ce composant est appelé *importation*. Lors de la définition d'une importation, vous devez spécifier la méthode d'accès à un service externe (emplacement), ainsi que le protocole d'appel.

Exportations

De même, si l'accès à votre composant s'effectue via des applications externes, ce qui est souvent le cas, vous devez le rendre accessible. Pour cela, utilisez un composant spécial qui affiche votre logique au "monde externe". Ce composant est appelé *exportation*. Les exportations peuvent être appelées de façon synchrone ou asynchrone.

Références autonomes

Dans WebSphere Process Server, un module de service SCA est intégré comme fichier EAR Java EE qui contient plusieurs autres sous-modules Java EE. Les éléments Java EE, tels qu'un fichier WAR, peuvent être intégrés au module SCA. Les artefacts autres que SCA, comme les JSP, peuvent également être intégrés à un module de service SCA. Ce conditionnement leur permet d'appeler des services SCA à l'aide du modèle de programmation client SCA grâce à un type de composant spécial appelé "référence autonome".

Le modèle de programmation SCA est hautement déclaratif. Les développeurs d'intégration peuvent configurer des aspects, comme le comportement transactionnel des appels, la propagation des données d'identification de sécurité, si un appel doit être synchrone ou asynchrone de façon déclarative, directement dans le diagramme d'assemblage. L'exécution SCA, non pas les développeurs, doit se charger de l'implémentation du comportement spécifié dans ces modificateurs. La flexibilité déclarative de SCA est l'une des fonctions les plus puissantes de ce modèle de programmation. Les développeurs peuvent se consacrer à implémenter la logique métier, au lieu de répondre aux aspects techniques, comme faciliter les mécanismes d'appel asynchrone. Tous ces aspects sont automatiquement gérés par l'exécution SCA.

Qualifiants

Les qualifiants régissent l'interaction entre un client de service et un service cible. Des qualifiants peuvent être spécifiés dans les références de composant de service, les interfaces et les implémentations. Ils sont généralement externes à une implémentation.

Les différentes catégories de qualifiants sont les suivantes :

- Transaction, qui spécifie la manière dont les contextes transactionnels sont gérés dans un appel SCA.
- Session d'activité, qui spécifie la manière dont les contextes de session d'activité sont propagés.
- Sécurité, qui spécifie les autorisations.

- La fiabilité asynchrone fournit des règles pour la distribution de messages asynchrones.

SCA autorise l'application de ces qualifiants de qualité de service (QoS - Quality of Service) aux composants de façon déclarative (sans aucune programmation ou modification du code d'implémentation des services). Vous pouvez ajouter des qualifiants de service à l'aide de WebSphere Integration Developer. Les qualifiants QoS sont généralement appliqués lorsque vous êtes prêt à envisager le déploiement d'une solution.

Concepts associés

«Modèle de programmation du client»
Le modèle de programmation du client SCA est conçu pour rechercher un service, créer des objets de données et gérer les exceptions générées par le composant appelé.

«Interfaces», à la page 143
Un composant de service est associé à une ou plusieurs interfaces. Les interfaces associées à un composant de service affichent les opérations métier associées à ce service.

«Styles d'appel», à la page 152
Avec SCA, vous pouvez appeler des composants de service à l'aide de styles de programmation synchrones et asynchrones. Vous pouvez assembler des modules dans des solutions globales où les canaux asynchrones entre les composants de service et les modules peuvent augmenter la capacité de traitement globale et la souplesse du système.

«Qualifiants», à la page 164
Les qualifiants représentent une partie importante de SCA car ils permettent aux développeurs de placer des exigences de qualité de service sur l'environnement d'exécution de WebSphere Process Server.

Tâches associées

«Développement de modules de service», à la page 145
Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

Modèle de programmation du client

Le modèle de programmation du client SCA est conçu pour rechercher un service, créer des objets de données et gérer les exceptions générées par le composant appelé.

Le modèle de programmation du client SCA offre deux fonctions principales aux clients. Le modèle de programmation expose une interface qui permet aux clients de rechercher des services dans le module actuel et, une fois que ce service a été détecté, le modèle de programmation du client permet au client d'appeler des opérations sur ce service.

Les clients recherchent les services à l'aide de la classe ServiceManager. Il existe plusieurs manières d'instancier la classe ServiceManager, en fonction de la portée de recherche souhaitée pour le service.

La principale interface que les clients doivent avoir à l'esprit pour la recherche des services est `com.ibm.websphere.sca.ServiceManager`. Cette interface inclut une méthode locateService qui renvoie une référence à l'implémentation de service du service demandé. Le paramètre de chaîne transmis dans la méthode locateService représente le nom de référence du service que le client souhaite rechercher. La

documentation Java du modèle de programmation du client SCA est incluse dans le centre de documentation de WebSphere Process Server, et est également incluse si vous choisissez d'installer la documentation Java dans le cadre de l'installation de WebSphere Process Server.

Une fois qu'un client a trouvé le service approprié, deux types de modèle d'appel peuvent être utilisés pour effectuer un appel à une opération ou une méthode offerte par le service. Tout d'abord, il existe un style d'interaction *appel dynamique*. L'interface clé de ce style d'interaction est `com.ibm.websphere.sca.Service`. La méthode invoke() de cette interface utilise le nom de l'opération que vous allez appeler, avec les paramètres requis, pour appeler cette opération.

```
public Interface MyService {
public String someMethod(String input);

Service myService = (Service) serviceManager.locateService("myService");
DataObject input = ...
DataObject result = (DataObject) myService.invoke("someMethod", input);
```

Les clients peuvent également utiliser une méthode d'*appel statique (type-safe)* pour appeler une opération particulière associée à un service. Ce type d'appel ne fonctionne que pour les définitions d'interface spécifiées comme définitions Java. Dans ce cas, le client distribue le retour de l'appel locateService() à l'interface appropriée et peut appeler les appels de méthode type safe appropriés sur cette interface.

```
public Interface MyService {
public String someMethod(String input);

MyService myService = (MyService) serviceManager.locateService("myService");
String input = ...
String result = myService.someMethod(input);
```

Interfaces

Un composant de service est associé à une ou plusieurs interfaces. Les interfaces associées à un composant de service affichent les opérations métier associées à ce service.

Tous les composants possèdent des interfaces de type WSDL. Seuls les composants Java prennent en charge les interfaces de type Java. Si un composant, une importation ou une exportation, possèdent plusieurs interfaces, toutes les interfaces doivent être identiques.

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:mfc="http://www.ibm.com/xmlns/prod/websphere/scdl/mfc/7.0.0"
        xmlns:ns1="http://HelloWorldLibrary/HelloWorld" xmlns:ns2="http://HelloService/HelloService"
        xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/7.0.0"
        Xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/7.0.0"
        displayName="HelloWorldMediation" name="HelloWorldMediation">
 <Interfaces>
  <interface xsi:type="wsdl:WSDLPortType" portType="ns1:HelloWorld">
   <scdl:interfaceQualifier xsi:type="scdl:JoinTransaction" value="true"/>
  </Interface>
 </Interfaces>
 <references>
  <reference name="HelloServicePartner">
   <interface xsi:type="wsdl:WSDLPortType" portType="ns2:HelloService"/>
   <scdl:referenceQualifier xsi:type="scdl:SuspendTransaction" value="false"/>
   <scdl:referenceQualifier xsi:type="scdl:Reliability"/>
   <scdl:referenceQualifier xsi:type="scdl:DeliverAsyncAt" value="commit"/>
   <wire target="HelloServiceImport"/>
  </Reference>
 </references>
 <implementation xsi:type="mfc:MediationFlowImplementation" mfcFile="HelloWorldMediation.mfc">
  <scdl:implementationQualifier xsi:type="scdl:Transaction" value="global"/>7
```

Figure 50. Exemple de définition d'interface dans une définition de composant SCDL

Ces interfaces peuvent être spécifiées comme interfaces Java ou interfaces de type de port WSDL. Toutefois, vous ne pouvez pas mélanger les interfaces Java et les interfaces de type de port WSDL sur la même définition de composant de service. Les arguments et types de retour de ces interfaces sont spécifiée comme des types Java simples, des classes Java, des objets de données de service ou un schéma XML (pour les interfaces de type de port WSDL).

Un composant peut être appelé de manière synchrone ou asynchrone, que l'implémentation soit synchrone ou asynchrone. Les interfaces de composant sont définies sous la forme synchrone et la prise en charge asynchrone est également générée pour elles. Vous pouvez spécifier un style d'interaction recommandé de type synchrone ou asynchrone. Le type asynchrone indique aux utilisateurs de l'interface qu'il contient au moins une opération dont l'exécution peut être assez longue. Par conséquent, le service appelant doit éviter de conserver une transaction ouverte lorsqu'il attend qu'une opération se termine et envoyer sa réponse. Le style d'interaction s'applique à toutes les opérations de l'interface.

Les composants de différents modules peuvent être connectés ensemble en publiant les services comme des exportations qui contiennent leurs interfaces et en faisant glisser les exportations dans le diagramme d'assemblage requis pour créer des importations. Lors de la connexion des composants, vous pouvez également spécifier des qualificateurs de qualité de service sur les implémentations, les références partenaires et les interfaces du composant.

Les importations possèdent des interfaces qui sont identiques à celles du service éloigné auquel elles sont associées ou représentent un sous-ensemble de ces interfaces, pour que ces services éloignés puissent être appelés. Les importations sont utilisées dans une application exactement de la même manière que les composants locaux. Il existe donc un modèle d'assemblage uniforme pour toutes les fonctions, quel que soit leur emplacement ou implémentation. Il n'est pas nécessaire de définir la liaison d'importation lors de la phase de développement ; elle peut l'être lors de la phase de déploiement.

Les exportations possèdent des interfaces qui sont identiques à celles du composant auquel elles sont associées ou représentent un sous-ensemble de ces

interfaces, pour que le service publié puisse être appelé. Une exportation déplacée d'un autre module dans un diagramme d'assemblage crée automatiquement une importation.

Développement de modules de service

Un composant de service doit être contenu dans un module de service. Le développement de modules destinés à contenir des composants est essentiel pour permettre la fourniture de services à d'autres modules.

task_prereq

Cette tâche suppose qu'une analyse des exigences montre que l'implémentation d'un composant de service que d'autres modules utiliseront est avantageuse.

task_context

Après avoir analysé vos besoins, vous pouvez décider que la fourniture et l'utilisation de composants de service constituent un moyen efficace de traiter les informations. Si vous déterminez que des composants de service réutilisables présenteraient un avantage pour votre environnement, créez un module de service destiné à contenir les composants de service.

task_procedure

1. Identifiez les composants de service que d'autres modules peuvent utiliser.

   Une fois que vous avez identifié les composants de service, passez à la section «Développement de composants de service».
2. Identifiez des composants de service dans une application qui pourraient utiliser des composants de service dans d'autres modules de service.

   Une fois que vous avez identifié les composants de service et les composants cibles correspondants, passez à la section «Appel de composants» ou «Appel dynamique des composants »
3. Reliez les composants client aux composants cible par le biais de connexions.

Concepts associés

Présentation du développement de modules
Un module est une unité de déploiement de base pour une application WebSphere Process Server. Un module peut contenir des composants, des bibliothèques et des modules de transfert utilisés par l'application.

Tâches associées

«Développement de composants de service», à la page 147
Développez des composants de service pour fournir une logique réutilisable à plusieurs applications dans votre serveur.

«Appel de composants», à la page 149
Les composants avec modules peuvent utiliser des composants sur n'importe quel noeud d'un cluster WebSphere Process Server.

«Appel dynamique d'un composant», à la page 151
Lorsqu'un module appelle un composant qui a une interface de type de port WSDL (Web Service Descriptor Language), il doit appeler le composant de façon dynamique à l'aide de la méthode invoke().

Présentation du développement de modules :

Un module est une unité de déploiement de base pour une application WebSphere Process Server. Un module peut contenir des composants, des bibliothèques et des modules de transfert utilisés par l'application.

Le développement de modules consiste notamment à assurer que les composants, les modules de transfert et les bibliothèques (collections d'artefacts référencés par le module) requis par l'application sont disponibles sur le serveur de production.

WebSphere Integration Developer est l'outil principal de développement des modules destinés à être déployés sur WebSphere Process Server. Bien qu'il soit possible de développer des modules dans d'autres environnements, il est préférable d'utiliser WebSphere Integration Developer.

WebSphere Process Server prend en charge les modules de service métier et les modules de médiation. Les deux modules et les modules de médiation sont des types de module SCA (Service Component Architecture). Un module de communication permet les communications entre applications en transformant l'appel de service dans un format compris par la cible, en transmettant la demande à la cible et en renvoyant le résultat au module émetteur. Un module de service métier implémente la logique d'un processus métier. Toutefois, un module peut également inclure la même logique de médiation qui peut être intégrée à un module de médiation.

Les sections suivantes décrivent comment implémenter et mettre à jour des modules sous WebSphere Process Server.

Composants

Les modules SCA contiennent des composants qui forment la structure de base permettant d'encapsuler la logique métier réutilisable. Les composants fournissent et consomment des services qui sont associés à des interfaces, des références et des implémentations. L'interface définit un contrat entre un composant de service et un composant appelant.

Avec WebSphere Process Server, un module peut soit exporter un composant de service pour qu'il soit utilisé par d'autres modules, soit importer un composant de service pour l'utiliser. Pour appeler un composant de service, un module appelant fait référence à l'interface du composant de service. Les références aux interfaces sont résolues à travers la configuration des références du module appelant à leurs interfaces respectives.

Pour développer un module, vous devez effectuer les activités suivantes :

1. Définir ou identifier des interfaces pour les composants du module.
2. Définir ou manipuler des objets métier utilisés par les composants.
3. Définir ou modifier des composants via leurs interfaces.

   Remarque : Un composant est défini par le biais de son interface.
4. OptionalColonSymbol Exporter ou importer des composants de service.
5. Créez un fichier d'archive d'entreprise (EAR - Enterprise Archive) à déployer dans la phase d'exécution. Créez le fichier à l'aide de la fonction EAR d'exportation dans WebSphere Integration Developer ou de la commande serviceDeploy.

Types de développement

WebSphere Process Server comprend un modèle de programmation de composant afin de faciliter un paradigme de programmation orientée services. Pour utiliser ce modèle, un fournisseur exporte les interfaces d'un composant de service de façon à ce qu'un client puisse importer ces interfaces et utiliser le composant de service comme s'il était local. Un développeur utilise soit des interfaces fortement typées, soit des interfaces dynamiquement typées pour implémenter ou appeler le composant de service. Les interfaces et leurs méthodes sont décrites dans la section Références de ce centre de documentation.

Après avoir installé des modules de service sur vos serveurs, vous pouvez utiliser la console d'administration pour modifier le composant cible pour une référence d'une application. La nouvelle cible doit accepter le même type d'objet métier et effectuer la même opération que ce que la référence de l'application demande.

Remarques concernant le développement de composants de service

Lorsque vous développez un composant de service, posez-vous les questions suivantes :

- Ce composant de service va-t-il être exporté et utilisé par un autre module ?

  Si c'est le cas, assurez-vous que la définition portant sur le composant va pouvoir être utilisée par un autre module.
- L'exécution de ce composant de service prendra-t-elle relativement longtemps ?

  Si c'est le cas, envisagez d'implémenter une interface asynchrone pour le composant de service.
- Est-ce avantageux de décentraliser le composant de service ?

  Si c'est le cas, envisagez de placer une copie du composant de service dans un module de service qui est déployé sur un cluster de serveurs afin de bénéficier du traitement parallèle.
- L'application nécessite-t-elle un mélange de ressources à une phase et à deux phases ?

  Si c'est le cas, assurez-vous d'activer le support du dernier participant pour l'application.

  Remarque : Si vous créez votre application à l'aide de WebSphere Integration Developer ou créez le fichier EAR installable à l'aide de la commande serviceDeploy, ces outils activent automatiquement le support pour l'application. Consultez la rubrique consacrée à l'«utilisation de ressources de validation à une phase et de ressources de validation à deux phases dans la même transaction» dans le centre de documentation de WebSphere Application Server Network Deployment.

Développement de composants de service :

Développez des composants de service pour fournir une logique réutilisable à plusieurs applications dans votre serveur.

task_prereq

Cette tâche suppose que vous avez déjà développé et identifié le traitement qui est utile pour plusieurs modules.

task_context

Plusieurs modules peuvent utiliser un composant de service. L'exportation d'un composant de service rend celui-ci disponible pour les autres modules qui se réfèrent à lui par le biais d'une interface. Cette tâche explique comment compiler le composant de service de manière à ce que d'autres modules puissent l'utiliser.

Remarque : Un composant de service unique peut contenir plusieurs interfaces.

task_procedure

1. Définir l'objet de données permettant de déplacer des données entre l'appelant et le composant de service.

   L'objet de données et son type font partie de l'interface entre les appelants et le composant de service.
2. Définir une interface que les appelants utiliseront pour référencer le composant de service.

   La définition de cette interface nomme le composant de service et répertorie toutes les méthodes disponibles dans ce composant de service.
3. Générer la classe implémentant l'appel du service.
4. Développer l'implémentation de la classe générée.
5. Sauvegarder les interfaces et les implémentations du composant dans des fichiers dotés d'une extension .java.
6. Empaqueter le module de service et les ressources nécessaires dans un fichier JAR.

   Reportez-vous à la section «Déploiement d'un module sur un serveur de production» de ce centre de documentation pour obtenir une description des étapes 6 à 8.
7. Exécuter la commande serviceDeploy pour créer un fichier EAR installable contenant l'application.
8. Installer l'application sur le noeud du serveur.
9. OptionalColonSymbol Configurer les connexions entre les appelants et le composant de service correspondant, en cas d'appel d'un composant de service d'un autre module de service.

   La section «Administration» de ce centre de documentation explique comment configurer ces connexions.

Exemples de développement de composants

Cet exemple montre un composant de service synchrone qui implémente une méthode unique, CustomerInfo. La première section définit l'interface du composant de service qui implémente une méthode appelée getCustomerInfo.

```
public interface CustomerInfo {
 public interface CustomerInfo { public Customer getCustomerInfo(String
customerID);
}
}
```

Le bloc de code suivant implémente le composant de service.

```
public class CustomerInfoImpl implements CustomerInfo {
 public Customer getCustomerInfo(String customerID) {
  Customer cust = new Customer();

  cust.setCustNo(customerID);
  cust.setFirstName("Victor");
```

```
  cust.setLastName("Hugo");
  cust.setSymbol("IBM");
  cust.setNumShares(100);
  cust.setPostalCode(10589);
  cust.setErrorMsg("");

  return cust;
 }
}
```

x

La section suivante est l'implémentation de la classe associée à StockQuote.

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

task_postreq

Appelez le service.

Appel de composants :

Les composants avec modules peuvent utiliser des composants sur n'importe quel noeud d'un cluster WebSphere Process Server.

task_prereq

Avant d'appeler un composant, assurez-vous que le module qui contient le composant est installé sur WebSphere Process Server.

task_context

Les composants peuvent utiliser n'importe quel composant de service disponible dans un cluster WebSphere Process Server en utilisant le nom du composant et en transférant le type de données qu'attend le composant. L'appel d'un composant dans cet environnement implique la localisation, puis la création de la référence vers le composant nécessaire.

Remarque : Un composant de module peut appeler un composant à l'intérieur du même modèle : cette opération s'appelle un appel intra-module. Implémentez les appels externes (appels inter-modules) en exportant l'interface dans le composant fournisseur et en important l'interface dans le composant appelant.

Important : Lors de l'appel d'un composant résidant sur un serveur autre que le serveur sur lequel s'exécute le module appelant, vous devez apportez des modifications de configuration à ces deux serveurs. Les configurations requises dépendent du mode d'appel du composant (appel asynchrone ou appel synchrone). La procédure de configuration spécifique des serveurs d'applications est décrite dans les tâches associées.

task_procedure

1. Déterminer les composants requis par le module appelant.

Notez le nom de l'interface dans un composant et le type de données dont l'interface a besoin.

2. Définir un objet de données.

   Bien que l'entrée ou le retour puisse être une classe Java, l'idéal est un objet de données de service.

3. Localiser le composant.

   a. Utiliser la classe ServiceManager pour obtenir les références disponibles pour le module appelant.

   b. Utiliser la méthode locateService() pour trouver le composant.

      En fonction du composant, l'interface peut être soit un type de port WSDL (Web Service Descriptor Language), soit une interface Java.

4. Appeler le composant de manière synchrone.

   Vous pouvez soit appeler le composant par le biais d'une interface Java, soit utiliser la méthode invoke() pour appeler le composant de manière dynamique.

5. Traiter le retour.

   Le composant peut générer une exception, aussi le client doit-il être capable de traiter cette possibilité.

Exemple d'appel d'un composant

L'exemple suivant permet de créer une classe ServiceManager.

```
ServiceManager serviceManager = new ServiceManager();
```

Cet exemple utilise la classe ServiceManager pour obtenir une liste de composants à partir d'un fichier contenant les références des de composants.

```
InputStream myReferences = new FileInputStream("MyReferences.references");
ServiceManager serviceManager = new ServiceManager(myReferences);
```

Le code suivant localise un composant qui implémente l'interface Java StockQuote.

```
StockQuote stockQuote = (StockQuote)serviceManager.locateService("stockQuote");
```

Le code suivant localise un composant qui implémente soit une interface Java, soit une interface de type de port WSDL. Le module appelant utilise l'interface Service afin d'interagir avec le composant.

Tip : Si le composant implémente une interface Java, il peut être appelé à l'aide de l'interface ou de la méthode invoke().

```
Service stockQuote = (Service)serviceManager.locateService("stockQuote");
```

L'exemple suivant illustre le code MyValue, qui appelle un autre composant.

```
public class MyValueImpl implements MyValue {

 public float myValue throws MyValueException {

  ServiceManager serviceManager = new ServiceManager();

     // variables
         Customer customer = null;
         float quote = 0;
         float value = 0;

     // invoke
         CustomerInfo cInfo = (CustomerInfo)serviceManager.locateService("customerInfo");
         customer = cInfo.getCustomerInfo(customerID);
```

```
     if (customer.getErrorMsg().equals("")) {

         // invoke
       StockQuote sQuote =
      (StockQuote)serviceManager.locateService("stockQuote");
       Ticket ticket =  sQuote.getQuote(customer.getSymbol());
    // ... do something else ...
       quote =  sQuote.getQuoteResponse(ticket, Service.WAIT);

         // assign
          value = quote * customer.getNumShares();
     } else {

         // throw
         throw new MyValueException(customer.getErrorMsg());
     }
     // reply
         return value;
 }
}
```

task_postreq

Configurez les connexions entre les références de module appelant et les interfaces de composant.

Appel dynamique d'un composant :

Lorsqu'un module appelle un composant qui a une interface de type de port WSDL (Web Service Descriptor Language), il doit appeler le composant de façon dynamique à l'aide de la méthode invoke().

task_prereq

Cette tâche suppose qu'un composant appelant appelle un composant de façon dynamique.

task_context

Avec une interface de type de port WSDL, un composant appelant doit utiliser la méthode invoke() pour appeler le composant. Un composant appelant peut également appeler un composant ayant une interface Java de cette façon.

task_procedure

1. Déterminez le module qui contient le composant nécessaire.
2. Déterminez le tableau dont le composant a besoin.

   Le tableau d'entrée peut être de l'un des trois types suivants :
   - Des types Java haut de casse primitifs ou des tableaux de ce type
   - Des classes Java ordinaires ou des tableaux de ces classes
   - Service Data Objects (SDO)
3. Définissez un tableau pour contenir la réponse du composant.

   Le tableau de réponse peut être des mêmes types que le tableau d'entrée.
4. Utilisez la méthode invoke() pour appeler le composant nécessaire et transférer l'objet tableau vers le composant.
5. Traitez le résultat.

Exemples d'appel dynamique d'un composant

Dans l'exemple suivant, un module utilise la méthode invoke() pour appeler un composant qui utilise des types de données Java haut de casse primitives.

```
Service service = (Service)serviceManager.locateService("multiParamInf");

  Reference reference = service.getReference();

  OperationType methodMultiType =
    reference.getOperationType("methodWithMultiParameter");

  Type t = methodMultiType.getInputType();

  BOFactory boFactory = (BOFactory)serviceManager.locateService
    ("com/ibm/websphere/bo/BOFactory");

  DataObject paramObject = boFactory.createbyType(t);

  paramObject.set(0,"input1")
  paramObject.set(1,"input2")
  paramObject.set(2,"input3")

  service.invoke("methodMultiParamater",paramObject);
```

L'exemple suivant utilise la méthode d'appel via une interface de port WSDL configurée en tant que cible.

```
Service serviceOne = (Service)serviceManager.locateService("multiParamInfWSDL");

 DataObject dob = factory.create("http://MultiCallWSServerOne/bos", "SameBO");
   dob.setString("attribute1", stringArg);

 DataObject wrapBo = factory.createByElement
  ("http://MultiCallWSServerOne/wsdl/ServerOneInf", "methodOne");
   wrapBo.set("input1", dob); //wrapBo encapsule tous les paramètres de methodOne
   wrapBo.set("input2", "XXXX");
   wrapBo.set("input3", "yyyy");

 DataObject resBo= (DataObject)serviceOne.invoke("methodOne", wrapBo);
```

Styles d'appel

Avec SCA, vous pouvez appeler des composants de service à l'aide de styles de programmation synchrones et asynchrones. Vous pouvez assembler des modules dans des solutions globales où les canaux asynchrones entre les composants de service et les modules peuvent augmenter la capacité de traitement globale et la souplesse du système.

Un composant expose les interfaces de niveau métier à la logique métier de son application pour que le service puisse être utilisé ou appelé. L'interface d'un composant définit les opérations qui peuvent être appelées et les données transmises, telles que les arguments en entrée, les valeurs renvoyées et les exceptions. Une importation et une exportation contient également des interfaces pour que le service publié puisse être appelé.

Tous les composants possèdent des interfaces de type WSDL. Seuls les composants Java prennent en charge les interfaces de type Java. Si un composant, une importation ou une exportation, possèdent plusieurs interfaces, toutes les interfaces doivent être identiques.

Un composant peut être appelé de manière synchrone ou asynchrone, que l'implémentation soit synchrone ou asynchrone. Les interfaces de composant sont définies sous la forme synchrone et la prise en charge asynchrone est également

générée pour elles. Vous pouvez spécifier un style d'interaction recommandé de type synchrone ou asynchrone. Le type asynchrone indique aux utilisateurs de l'interface qu'il contient au moins une opération dont l'exécution peut être assez longue. Par conséquent, le service appelant doit éviter de conserver une transaction ouverte lorsqu'il attend qu'une opération se termine et envoyer sa réponse. Le style d'interaction s'applique à toutes les opérations de l'interface.

Lors de la création d'applications dans WebSphere Integration Developer, il est recommandé de définir explicitement le style d'appel utilisé par chacun de vos composants pour s'appeler entre eux. Vous devez au moins connaître les styles d'appel utilisés dans votre application lorsque vous analysez les performances ou développez votre stratégie de traitement des erreurs. Vous devez bien sûr comprendre les interactions dans votre application lorsque vous étudiez/définissez les limites de vos transactions. Les utilisateurs sont souvent surpris de découvrir que la définition ou la détermination des styles d'appel entre les composants n'est pas une tâche aussi facile qu'elle paraît. Cette section explique comment définir ou déterminer le style d'appel utilisé lors de la phase d'exécution, en fonction de caractéristiques spécifiques de votre application.

Les styles d'appel fournis pas SCA sont les suivants :

- Synchrone
- Asynchrone avec opération unidirectionnelle
- Asynchrone avec rappel
- Asynchrone avec réponse différée

API SCA utilisée dans l'implémentation Java pour que l'appel détermine le style d'appel lors de la phase d'exécution :

- `invoke()` : Synchrone
- `invokeAsync()` : Asynchrone
- `invokeAsyncWithCallback()` : Asynchrone

En général, lorsqu'il étudie une interaction d'un composant (source ou client) vers un autre (cible), le client du service détermine le type d'appel utilisé. Par exemple, si votre composant source est un composant Java, le style d'appel de la source à la cible est déterminé par l'API d'appel SCA particulière que vous utilisez dans l'implémentation, telle qu'invoke(), invokeAsync() ou invokeAsyncWithCallback(). Chacun des autres composants fournis dans WebSphere Process Server contient un ensemble de règles qu'il utilise pour déterminer si un appel est synchrone ou asynchrone.

Certains composants/Certaines importations sont considérés comme asynchrones :

- BPEL de longue durée
- Tâches manuelles
- MQ/Importations MQ
- JMS/Importations génériques
- JMS/Importations JMS

Tous les appels à des composants ou des importations de ces types doivent correspondre à des appels asynchrones. Si le composant appelant (ou source) lance l'interaction de manière synchrone, SCA la transforme en interaction asynchrone.

Concepts associés

«Appel synchrone»
Les interfaces SCA (interfaces de composant de service) sont toujours définies sous la forme synchrone. Pour chaque interface synchrone, une ou plusieurs interfaces asynchrones peuvent être générées.

«Appel asynchrone», à la page 155
WebSphere Process Server offre un puissant modèle de programmation pour le développement d'applications asynchrones. Avec l'appel asynchrone dans SCA, il existe trois types de style d'interaction asynchrone disponibles : unidirectionnelle, réponse différée et demande avec appel renouvelé. Avec ces trois types d'appel asynchrone, le client reçoit de nouveau le contrôle immédiatement de l'environnement d'exécution SCA lors d'un appel `invokeAsync()`.

«Interactions SCA», à la page 156
SCA prend en charge les appels de module synchrones et asynchrones. Les développeurs ont la possibilité de sélectionner les interfaces et méthodes d'appel appropriées pour leurs interactions SCA.

«Traitement des exceptions d'un appel synchrone», à la page 157
Si un composant de service est appelé de manière synchrone, le client et le fournisseur de services sont exécutés dans la même unité d'exécution. La cible peut renvoyer un message de réponse, une exception ou rien (dans une opération unidirectionnelle) au client. Si le résultat est une exception, il peut s'agir d'une exception métier ou d'une exception système. Dans ce cas, le client peut correspondre à du code d'application ou à une forme de code système.

«Traitement des exceptions d'un appel asynchrone», à la page 158
Si un composant de service est appelé de manière asynchrone, le client et le fournisseur de services sont exécutés dans des unités d'exécution différentes et des conditions d'erreur peuvent se produire dans l'une ou l'autre de ces unités d'exécution. Le client peut rencontrer une exception système lors de l'appel ou le fournisseur de services peut rencontrer une exception métier ou système lorsqu'il traite la demande.

«Considérations à prendre en compte lors de l'appel de services sur des serveurs différents», à la page 159
L'un des avantages offerts par l'architecture orientée services est la possibilité pour les consommateurs d'utiliser des services qui se trouvent dans d'autres modules de service. Pour équilibrer équitablement la charge de travail, vous pouvez installer les applications sur des serveurs différents d'une cellule et ces applications peuvent résider sur des serveurs physiques différents.

Appel synchrone :

Les interfaces SCA (interfaces de composant de service) sont toujours définies sous la forme synchrone. Pour chaque interface synchrone, une ou plusieurs interfaces asynchrones peuvent être générées.

Si un composant de service est appelé de manière synchrone, le client (consommateur) et le fournisseur de services sont exécutés dans la même unité d'exécution. Le composant appelant dans WebSphere Process Server est bloqué jusqu'à la réception d'une réponse du fournisseur.

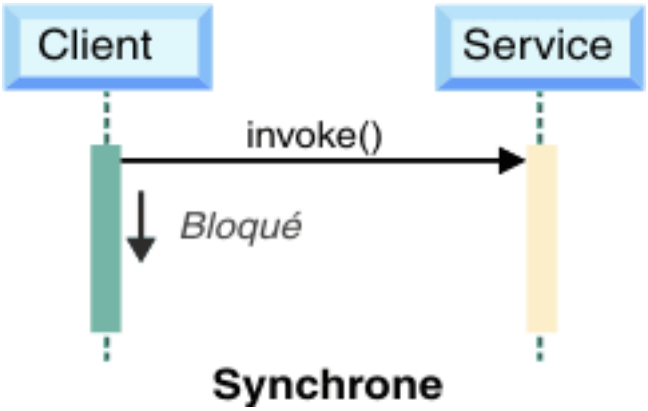


Figure 51. Appel synchrone

Appel asynchrone :

WebSphere Process Server offre un puissant modèle de programmation pour le développement d'applications asynchrones. Avec l'appel asynchrone dans SCA, il existe trois types de style d'interaction asynchrone disponibles : unidirectionnelle, réponse différée et demande avec appel renouvelé. Avec ces trois types d'appel asynchrone, le client reçoit de nouveau le contrôle immédiatement de l'environnement d'exécution SCA lors d'un appel `invokeAsync()`.

En plus de ses API publiées et outils permettant de développer des programmes synchrones à l'aide de Java, WebSphere Process Server intègre un certain nombre de liaisons de messagerie asynchrones et de composants asynchrones.

Le client a le choix entre trois méthodes pour capturer la réponse ultérieurement. Il peut choisir de supprimer la réponse intégralement ou s'il s'agit d'un appel à une méthode vide. Dans ce cas, l'appel asynchrone est dit *unidirectionnel*. Une autre option pour le client consiste à appeler `invokeAsync()`, puis à continuer le traitement jusqu'à ce qu'il demande la capture de la réponse. Ce scénario est appelé *réponse différée*. Enfin, le client peut également effectuer une *demande avec rappel* asynchrone. Pour cela, le client doit d'abord implémenter l'interface ServiceCallback. Ensuite, après avoir appelé `invokeAsync()`, l'environnement d'exécution SCA fournit un rappel au gestionnaire ServiceCallback pour fournir la réponse au client.

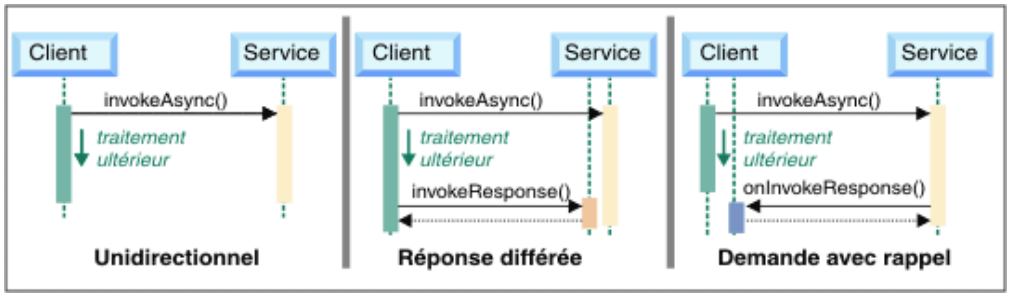


Figure 52. Modèles d'appel asynchrone

Les interfaces SCA sont toujours définies sous la forme synchrone. Pour chaque interface synchrone, une ou plusieurs interfaces asynchrones peuvent être générées. Lorsqu'un mécanisme de rappel est choisi par le client, le composant client doit

implémenter une classe : <nom de l'interface>.Callback.java. L'interface de cette classe découle de l'interface du composant que le client souhaite utiliser.

Interactions SCA :

SCA prend en charge les appels de module synchrones et asynchrones. Les développeurs ont la possibilité de sélectionner les interfaces et méthodes d'appel appropriées pour leurs interactions SCA.

Le diagramme récapitule les différents types d'interface, les méthodes d'appel et modèles pris en charge et la manière dont les données sont transmises entre le client et le service.

Pour un appel synchrone, les données sont transmises par référence dans le même module SCA, alors que pour les appels asynchrones, les données sont transmises par valeur. Le tableau récapitule également quand il est possible d'utiliser un appel dynamique ou sécurisé du type en fonction du type d'interface. Les méthodes d'appel dynamique sont toujours disponibles pour un type de port WSDL ou des interfaces Java. Toutefois, pour que des méthodes d'appel sécurisées soient disponibles pour le client, un type d'interface Java doit être utilisé pour la définition d'interface sur la référence client appropriée.

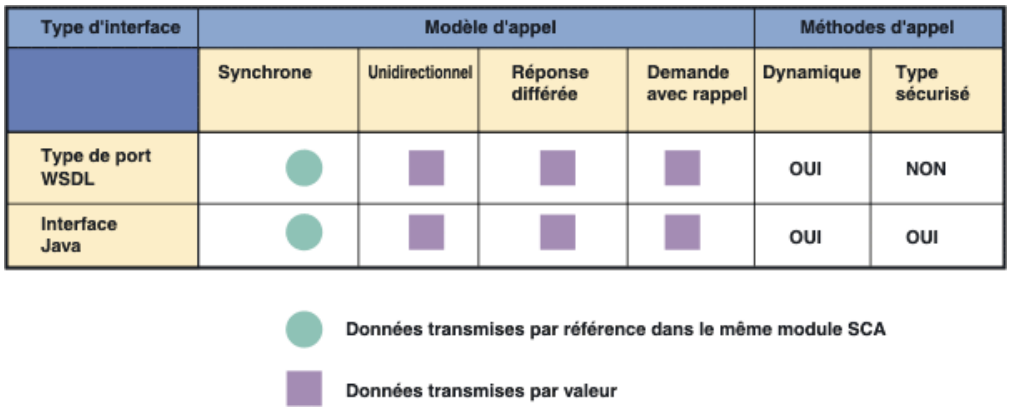


Figure 53. Récapitulatif des modèles d'appel avec les méthodes prises en charge pour la transmission de données

Appel dynamique des clients

Il existe un certain nombre d'interfaces et de méthodes clé requises pour prendre en charge les interactions synchrone et asynchrone lors de l'utilisation de l'appel dynamique du client.

Tableau 15. Récapitulatif des interfaces et des méthodes clé pour l'appel dynamique du client

Interface	Méthodes	Description
Maintenance	Object invoke(Service String, Object)	Permet d'appeler des demandes de service synchrones
	Ticket invokeAsync(String, Object)	Permet d'appeler des demandes de service asynchrones **unidirectionnelles** ou **à réponse différée**
	Ticket invokeAsyncWithCallback(String, Object)	Permet d'appeler des demandes de service asynchrones de type **demande avec rappel**. Le client doit implémenter l'interface ServiceCallback
	Object invokeResponse(Ticket, long)	Permet d'obtenir une réponse dans le cas d'un appel **à réponse différée**
ServiceCallback	void onInvokeResponse(Ticket, Object, Exception)	L'interface de rappel doit être implémentée par le client à l'aide d'un appel de service asynchrone de type **demande avec rappel**

Traitement des exceptions d'un appel synchrone :

Si un composant de service est appelé de manière synchrone, le client et le fournisseur de services sont exécutés dans la même unité d'exécution. La cible peut renvoyer un message de réponse, une exception ou rien (dans une opération unidirectionnelle) au client. Si le résultat est une exception, il peut s'agir d'une exception métier ou d'une exception système. Dans ce cas, le client peut correspondre à du code d'application ou à une forme de code système.

Voici un exemple de client qui appelle un composant Java déclaré avec une interface JType. L'interface contient une méthode déclarée comme suit :

```
public interface StockQuote {
 float getQuote(String symbol) throws InvalidSymbolException;
}
```

Le code du client se présente comme suit :

```
try {
 float quote = StockQuoteService.getQuote(String symbol);
 } catch (InvalidSymbolException s) {
 System.out.println(Exception métier déclarée dans l'interfacecJava.);
 } catch (ServiceRuntimeException e) {
 System.out.println(Exception système non vérifiée détectée);
}
```

Dans ce scénario, la première exception InvalidSymbolException indique que la demande a atteint le fournisseur de services, qui ne reconnaît pas l'entrée du client. Le fournisseur de services génère ensuite une exception métier indiquant que le symbole fourni n'est pas valide. Cette exception métier est la seule déclarée par la signature de méthode.

Les exceptions JType, telle qu'InvalidSymbolException ne sont interceptées qu'avec les clients qui utilisent une référence JType.

Outre l'exception métier déclarée, le client peut recevoir des exceptions système. Par exemple, si le système boursier rencontre une difficulté, le service risque de ne pas obtenir la cotation et de générer des exceptions non vérifiées. Lorsqu'une exception de ce type est générée par le service, une exception ServiceRuntimeException est renvoyée au client et ce dernier peut alors souhaiter en déterminer le motif sous-jacent. Le fragment de code suivant indique comment il peut obtenir ces informations :

```
try {
 float quote = StockQuoteService.getQuote(String symbol);
 } catch (ServiceRuntimeException e) {
 Throwable t = e.getCause();
 if (t instanceof RemoteException) {
 system.out.println(System ran into RemoteException. Details as follows:  + e.toString());
}
```

Traitement des exceptions d'un appel asynchrone :

Si un composant de service est appelé de manière asynchrone, le client et le fournisseur de services sont exécutés dans des unités d'exécution différentes et des conditions d'erreur peuvent se produire dans l'une ou l'autre de ces unités d'exécution. Le client peut rencontrer une exception système lors de l'appel ou le fournisseur de services peut rencontrer une exception métier ou système lorsqu'il traite la demande.

Dans WebSphere Process Server, il existe toujours un équivalent asynchrone de l'interface synchrone.

Voici un exemple d'interface asynchrone :

```
public interface StockQuoteAsync {
public Ticket getQuoteAsync(String arg0);
public Ticket getQuoteAsyncWithCallback(String arg0);
public float getQuoteResponse(Ticket ticket, long timeout) throws InvalidSymbolException;
}
```

Voici le code client d'un appel à l'aide du modèle d'appel d'une réponse différée :

```
Ticket ticket = stockQuote.getQuoteAsync(symbol);
try {
 quote = stockQuote.getQuoteResponse(ticket, Service.WAIT);
 } catch (InvalidSymbolException s) {
 System.out.println(Exception métier déclarée dans l'interface.);
 } catch (ServiceRuntimeException e) {
 System.out.println(Exception système non vérifiée détectée);
}
```

Comme un appel synchrone, l'exceptionInvalidSymbolException indique que la demande a atteint le fournisseur de services, qui a généré une exception métier indiquant que le symbole n'était pas valide. Cette exception métier est la seule déclarée par l'interface. Les exceptions JType, telle qu'InvalidSymbolException ne sont interceptées qu'avec les clients qui utilisent une référence JType.

Outre l'exception métier déclarée, le client peut recevoir des exceptions système, telles qu'une erreur de connexion lors de l'envoi du message. Le client ne peut pas recevoir d'exceptions système qui se produisent dans l'unité d'exécution du service (unité d'exécution du côté service de l'appel asynchrone). Conformément au modèle de programmation asynchrone de SCA, les exceptions d'exécution qui se produisent sur le composant cible ne sont pas renvoyées au composant source.

Exception dans le traitement des exceptions asynchrones

Il existe une exception à la règle du modèle de programmation asynchrone de SCA : les exceptions d'exécution qui se produisent sur le composant cible ne sont pas renvoyées au composant source. Si le composant source est un composant de processus métier ou de processus Staff, les exceptions système qui se produisent dans le composant de service cible sont renvoyés à l'appelant. Cette fonctionnalité permet aux concepteurs de processus métier de modéliser et d'intercepter les exceptions systèmes, mais aussi d'exécuter la logique d'erreur si un client BPEL renvoie une exception système.

Considérations à prendre en compte lors de l'appel de services sur des serveurs différents :

L'un des avantages offerts par l'architecture orientée services est la possibilité pour les consommateurs d'utiliser des services qui se trouvent dans d'autres modules de service. Pour équilibrer équitablement la charge de travail, vous pouvez installer les applications sur des serveurs différents d'une cellule et ces applications peuvent résider sur des serveurs physiques différents.

L'un des avantages de WebSphere Process Server est la possibilité de répartir la charge de travail des applications entre plusieurs serveurs d'une cellule. Cette distribution permet un meilleur équilibrage de la charge de travail entre les divers serveurs de la cellule et optimise la facilité de maintenance des ressources informatiques car il n'existe qu'une copie d'une application ou d'un service sur le serveur. Ainsi, une application sur le serveur A peut demander un service installé sur le serveur B de la cellule. Pour utiliser les services de cette manière, vous devez configurer les communications entre les serveurs. Le type de configuration que vous effectuez varie suivant que le composant du service appelant appelle le service de manière asynchrone ou synchrone.

Des rubriques connexes décrivent comment configurer les systèmes pour des appels asynchrones et synchrones.

Tâches associées

«Configuration des serveurs pour qu'ils appellent des services de manière asynchrone»
Pour permettre à des composants de service installés sur des serveurs différents de communiquer, vous devez configurer ces derniers de manière similaire. Cette rubrique décrit la configuration à effectuer pour que des applications qui appellent de manière asynchrone des services installés sur un serveur différent puissent communiquer.

«Configuration des serveurs pour qu'ils appellent des services de manière synchrone», à la page 162
Lorsqu'un composant de service appelle un autre composant de service de manière synchrone, vous devez configurer le composant de service appelant de manière à ce qu'il pointe vers le système qui exécute la cible afin que le service cible puisse communiquer les résultats au composant de service appelant.

Configuration des serveurs pour qu'ils appellent des services de manière asynchrone :

Pour permettre à des composants de service installés sur des serveurs différents de communiquer, vous devez configurer ces derniers de manière similaire. Cette rubrique décrit la configuration à effectuer pour que des applications qui appellent de manière asynchrone des services installés sur un serveur différent puissent communiquer.

task_prereq

Pour exécuter cette tâche, vous devez avoir déjà installé WebSphere Process Server sur les systèmes dont vous configurez les communications, mais pas encore installé les applications concernées. Vous utilisez une console d'administration capable de consulter et de modifier la configuration des deux serveurs impliqués.

task_context

Avant d'installer une application pour laquelle un composant de service installé sur un autre système est requis, vous devez configurer les systèmes de manière à ce qu'ils puissent communiquer les demandes. Dans le cas des modules de service qui utilisent des appels asynchrones, le processus inclut la mise en oeuvre de médiations de bus externes et de bus d'intégration de service (SIB).

Remarque : Dans le cadre de l'exécution de cette tâche, le module de service appelant se trouve sur le système A et la cible sur le système B.

Dans le cadre de l'exécution de cette tâche, la figure 54 contient les informations à utiliser dans la configuration.

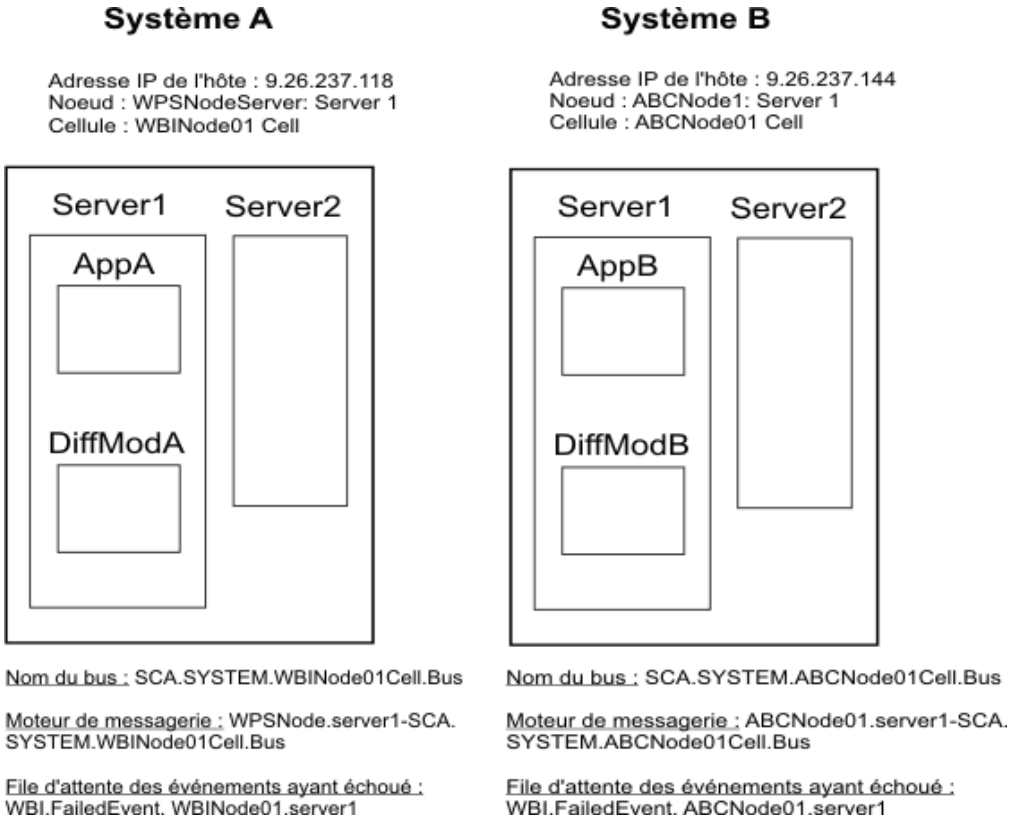


Figure 54. Appel d'un service sur un système différent

Remarque : Pour plus de simplicité, seuls sont visibles les serveurs impliqués dans cette communication dans chaque cellule et chaque serveur réside sur une machine physique distincte.

task_procedure

1. Collectez les informations sur chaque serveur impliqué dans la communication. Vous avez besoin des informations suivantes à la fois pour le serveur émetteur et le serveur cible :
   - Adresse IP de l'hôte
   - Cellule
   - Noeud
   - Serveur
   - Nom du bus
   - Moteur de messagerie
   - Nom de la file d'attente d'événements ayant échoué
2. Installez les applications.
3. Créez un bus externe sur chaque serveur désignant l'autre serveur et définissez le type de définition de routage sur `Direct, service integration bus link`.

   Pour plus d'informations, voir Connecting service integration buses to use point-to-point messaging dans le centre de documentation WebSphere Application Server Network Deployment, version 7.

   Dans le présent exemple, la configuration du lien de médiation du bus externe et du bus SIB sur le système A correspond à :

   ```
   Name of Service integration bus to connect to (the foreign bus):
   SCA.SYSTEM.ABCNode01Cell.Bus
   Gateway messaging engine in the foreign bus:
   ABCNode01.server1-SCA.SYSTEM.ABCNode01Cell.Bus
   Service integration bus link name: TestCrossCell
   Bootstrap service integration bus provider endpoints:
   9.26.237.144:7277:BootstrapBasicMessaging
   ```

   La configuration du lien de médiation du bus externe et du bus SIB sur le système B correspond à :

   ```
   Name of Service integration bus to connect to (the foreign bus):
   SCA.SYSTEM.WBINode01Cell.Bus
   Gateway messaging engine in the foreign bus:
   WPSNode.server1-SCA.SYSTEM.WBINode01Cell.Bus
   Service integration bus link name: TestCrossCell
   Bootstrap service integration bus provider endpoints:
   9.26.237.118:7276:BootstrapBasicMessaging
   ```

   Attention : Le numéro de port dans l'amorçage correspond au port d'adresse du noeuf final SIB. Si vous avez activé la sécurité, vous devez utiliser le port d'adresse du noeud final SIB sécurisé.
4. Synchronisez les liens de médiation du bus SIB en redémarrant les serveurs.

   Des messages similaires à ceux-ci s'affichent :

   ```
   [9/25/09 8:04:23:406 CDT] 00000034 SibMessage I [:] CWSIT0032I:
   The service integration bus link TestCrossCell from messaging engine
   WPSNode01.server1-SCA.SYSTEM.WPSNode01Cell.Bus in bus SCA.SYSTEM.WPSNode01Cell.Bus
   to messaging engine ABCNode01.server1-SCA.SYSTEM. ABCNode01Cell.Bus in bus SCA.SYSTEM.
   ABCNode01Cell.Bus started.
   ```
5. Affichez les destinations de chaque module de service.
6. Modifiez le chemin de transfert par défaut des destinations sortantes du module de service appelant qui doit être établi vers les cibles sur l'autre système.

   Sélectionnez **Applications > Modules SCA**, choisissez le module, puis cliquez sur **Destinations du bus système SCA**.

   La destination avec laquelle établir un lien est celle dont le nom contient `importlink`. Par exemple, sur le système A, la destination correspond à `sca/AppA/importlink/test/sca/cros/simple/custinfo/CustomerInfo`. Modifiez

le chemin en ajoutant comme préfixe au nom de la destination le nom du bus externe. Dans le présent exemple, le nom du bus externe du deuxième système correspond à `SCA.SYSTEM.ABCNode01Cell.Bus`. Le résultat est le suivant :

```
SCA.SYSTEM.ABCNode01Cell.Bus:sca/AppA/importlink/
test/sca/cros/simple/custinfo/CustomerInfo
```

7. OptionalColonSymbol Ajoutez les rôles expéditeur aux bus externes si vous avez activé la sécurité sur les systèmes. Veillez à définir l'utilisateur utilisé par chaque application sur les deux systèmes à partir de l'invite de commande du système d'exploitation. La commande permettant d'ajouter le rôle est la suivante :

   ```
   wsadmin $AdminTask addUserToForeignBusRole -bus nomBus
     -foreignBus nomBusExterne -role nomRôle -user nomUtilisateur
   ```

   Où :

   nomBus
   Nom du bus sur le système sur lequel vous entrez la commande.

   nomBusExterne
   Bus externe auquel vous ajoutez l'utilisateur.

   nomUtilisateur
   ID utilisateur à ajouter au bus externe.

task_postreq

Démarrez les applications.

Configuration des serveurs pour qu'ils appellent des services de manière synchrone :

Lorsqu'un composant de service appelle un autre composant de service de manière synchrone, vous devez configurer le composant de service appelant de manière à ce qu'il pointe vers le système qui exécute la cible afin que le service cible puisse communiquer les résultats au composant de service appelant.

task_prereq

Pour exécuter cette tâche, vous devez avoir déjà installé WebSphere Process Server sur les systèmes dont vous configurez les communications, mais pas encore installé les applications concernées. Vous utilisez une console d'administration capable de consulter et de modifier la configuration des deux serveurs impliqués.

task_context

Un composant de service qui appelle un autre service de manière synchrone ne peut communiquer avec la cible que si le nom d'exportation JNDI (Java Naming and Directory Interface) du système cible est configuré sur un nom JNDI du système appelant.

Remarque : Dans le cadre de l'exécution de cette tâche, le module de service appelant se trouve sur le système A et la cible sur le système B.

Dans le cadre de l'exécution de cette tâche, la figure 55, à la page 163 contient les informations à utiliser dans la configuration.

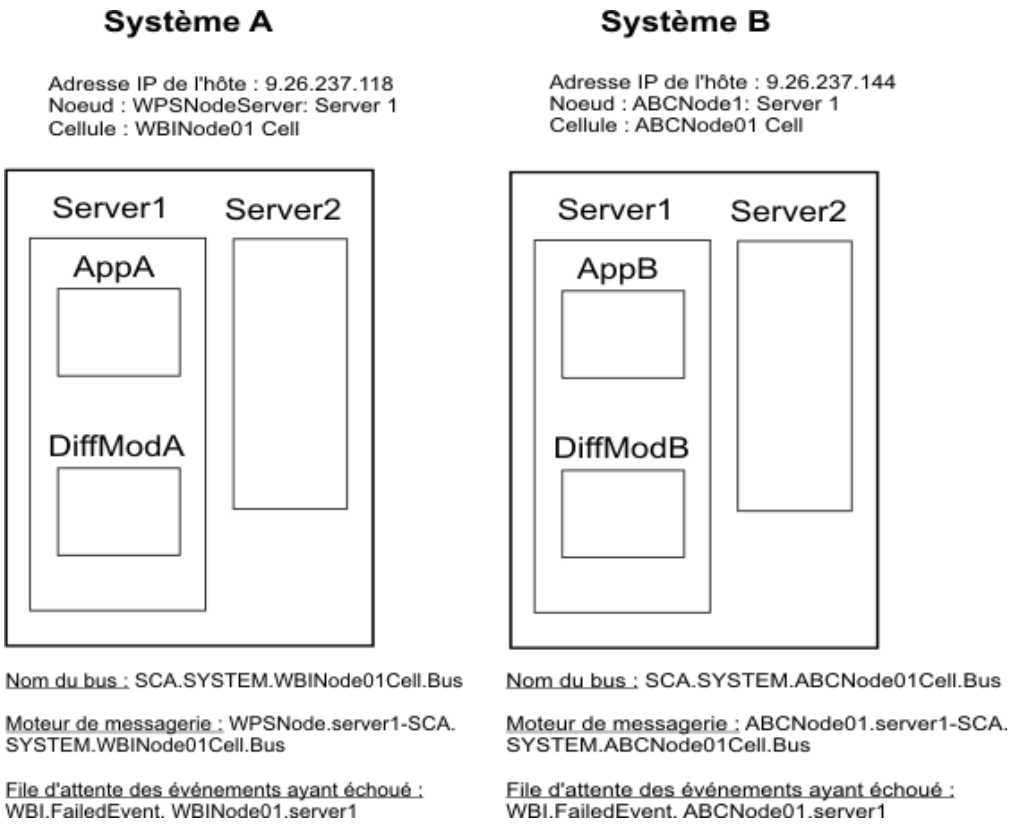


Figure 55. Appel d'un service sur un système différent

Remarque : Pour plus de simplicité, seuls sont visibles les serveurs impliqués dans cette communication dans chaque cellule et chaque serveur réside sur une machine physique distincte.

task_procedure

1. Installez les applications sur chaque serveur.
2. Créez une nouvelle liaison d'espace de nom sur le système appelant (système A, dans le présent exemple) qui pointe vers l'exportation sur le système cible.

   Dans le panneau **Liaisons d'espace de nom**, sélectionnez une portée de cellule et cliquez sur **Appliquer**. Une fois la portée modifiée, cliquez sur **Nouveau** pour créer la nouvelle liaison.

   Dans l'assistant, spécifiez les éléments suivants (les valeurs sont adaptées pour la configuration présentée en exemple) :

   a. Le type de liaison correspond à CORBA
   b. Les propriétés de base correspondent à :
      - L'identificateur de liaison est une chaîne unique, dans le présent exemple : `sca_import_test_sca_cross_simple_custinfo_CustomerInfo`
      - Le nom de l'espace de nom correspond au nom JNDI du bean que vous appelez sur le système cible, par exemple :

        `sca/AppB/export/test/sca/cros/simple/custinfo/CustomerInfo`

        Ce nom désigne l'interface d'exportation sur le système cible.
      - L'adresse URL Corbaname correspond à l'adresse IP et au numéro de port du service de nommage sur le système cible :

        `corbaname:iiop:`*`hôte:port/NameServiceServerRoot#<package.qualified.interface>`*

Remarque : Pour WebSphere Process Server, le port correspond à `BOOTSTRAP_ADDRESS`.

Exemple :

```
corbaname:iiop:9.26.237.144:2809/NameServiceServerRoot#sca/
  AppB/export/test/sca/cros/simple/custinfo/CustomerInfo
```

Lorsque vous avez terminé, cliquez sur **Suivant** et vérifiez les valeurs dans la page **Récapitulatif**. Après avoir effectué les vérifications nécessaires, cliquez sur **Terminer**.

La nouvelle liaison apparaît sur votre système.

3. Sauvegardez les modifications en cliquant sur **Sauvegarder**.
4. Si la configuration intercellule est composée de serveurs sur le même hôte du même nom et si des erreurs de recherche JNDI générant une exception `NameNotFoundException` se produisent, vous devez définir une propriété système.

   Suivez les instructions fournies dans Application access problems sous l'en-tête *Two servers with the same name running on the same host are being used to interoperate*.

task_postreq

Démarrez les applications. Le composant de service sur le système A peut maintenant appeler de manière synchrone le service sur le système B.

Qualifiants

Les qualifiants représentent une partie importante de SCA car ils permettent aux développeurs de placer des exigences de qualité de service sur l'environnement d'exécution de WebSphere Process Server.

Plusieurs catégories de qualifiants sont disponibles dans SCA. Il s'agit des catégories suivantes : transaction, session d'activité, sécurité et fiabilité asynchrone.

Chaque qualifiant SCA possède une portée particulière dans la définition SCDL d'un composant ou le qualifiant peut être spécifié. Par exemple, certains qualifiants peuvent être spécifiés au niveau de la référence, tandis que d'autres peuvent n'être valides qu'au niveau des interfaces ou de l'implémentation. Le tableau ci-après répertorie les divers qualifiants disponibles et la portée valide de chacun d'eux. Les qualifiants sont triés en fonction du type de qualité de service qu'ils offrent (par exemple, transaction ou sécurité).

Tableau 16. Récapitulatif des qualifiants

Type	Qualifiant	Portée	Description
Transaction	transaction	Implémentation	**global** – Une transaction globale doit être présente pour exécuter le composant **local** – Une transaction globale ne doit pas être présente pour exécuter le composant **any** – Le composant n'est pas affecté par l'état transactionnel **local application** – Le composant est traité au sein d'une transaction locale WebSphere gérée par l'application
	joinTransaction	Interface	**true** – Le conteneur d'hébergement rejoint la transaction client **false** (valeur par défaut) – Le conteneur d'hébergement ne rejoint pas la transaction client
	suspendTransaction	Référence	**true** – Les appels synchrones du composant cible ne sont pas exécutés dans la transaction globale du client. **false** – Les appels synchrones du composant cible sont exécutés dans la transaction globale du client
	deliverAsyncAt	Référence	**call** – Les appels asynchrones d'un service cible sont effectués immédiatement **commit** – Les appels asynchrones d'un service cible sont effectués dans le cadre d'une transaction globale

Tableau 16. Récapitulatif des qualifiants (suite)

Type	Qualifiant	Portée	Description
Réponse asynchrone	reliability	Référence	Spécifie le niveau de qualité de service pour la distribution des messages asynchrones. La fiabilité peut prendre l'une des deux valeurs suivantes : **bestEffort** ou **assured**
	requestExpiration	Référence	Spécifie le délai (en millisecondes) après lequel une demande asynchrone doit être supprimée si elle n'est pas distribuée
	responseExpiration	Référence	Spécifie la durée (en millisecondes) entre le moment où une demande est envoyé et celui où une réponse ou un rappel est reçu
Sécurité	securityIdentity	Implémentation	Le **droit** spécifie un nom logique pour l'ID sous lequel l'implémentation est exécutée lors de la phase d'exécution.
	securityPermission	Interfaces, Interface, Méthode	L'identité de l'appelant doit posséder le rôle spécifié par ce qualifiant pour disposer des droits permettant d'exécuter l'interface ou la méthode

Tableau 16. Récapitulatif des qualifiants (suite)

Type	Qualifiant	Portée	Description
Session d'activité	activitySession	Implémentation	**true** – Une session d'activité doit être établie pour exécuter ce composant **false** – Le composant est exécuté sans session d'activité**any** – le composant ne dépend pas de la présence ou de l'absence d'une session d'activité
	joinActivitySession	Interface	**true** – Le conteneur d'hébergement rejoint la session d'activité client **false** – Le conteneur d'hébergement ne rejoint pas la session d'activité client
	suspendActivitySession	Référence	**true** – Les méthodes du composant cible NE SONT PAS exécutées dans le cadre d'une session d'activité client **false** – Les méthodes du composant cible sont exécutées dans le cadre d'une session d'activité client

Qualifiants de transaction

Les qualifiants de transaction permettent aux développeurs de demander un environnement transactionnel particulier pour les composants de service d'un module SCA. Voici un récapitulatif de ces qualifiants :

transaction
Le qualifiant transaction est défini au niveau de l'implémentation d'un composant de service. Ce qualifiant accepte les valeurs 'global', 'local' (valeur par défaut) et 'any'. Si la valeur est global, le composant est exécuté dans le contexte d'une transaction globale. Si une transaction globale est présente lors de l'appel, le composant est ajouté à cette portée de transaction globale. Si la valeur est local, le composant est exécuté dans le contexte d'une transaction locale. Enfin, lorsque la valeur est any, si une transaction globale est présente, le composant rejoint la portée de la transaction globale actuelle. Toutefois, si aucune transaction globale n'est présente, le composant est exécuté dans le contexte d'une transaction locale.

joinTransaction
Le qualifiant joinTransaction est défini au niveau de l'interface d'un

composant de service. Ce qualifiant accepte les valeurs true et false (valeur par défaut). Si la valeur est true, l'environnement d'exécution n'interrompt pas une transaction globale (s'il en existe une) à la limite de l'interface. Si la valeur est false, l'environnement d'exécution interrompt une transaction globale (s'il en existe une) à la limite de l'interface. Le placement du qualifiant transactionnel joinTransaction sur une interface fournit des métadonnées qui permettent aux assembleurs et déployeurs de s'assurer que l'application assemblée se comporte normalement. C'est à l'assembleur ou au déployeur, avec les environnement d'exécution dynamiques, de déterminer si un composant cible est fédéré avec une transaction propagée.

suspendTransaction
Le qualifiant suspendTransaction est défini au niveau de la référence d'un composant de service et indique si une transaction globale doit être suspendue avant l'appel du service cible associé à la référence. Ce qualifiant accepte les valeurs true et false (valeur par défaut).

deliveryAsyncAt
Le qualifiant deliveryAsyncAt est similaire au qualifiant suspendTransaction, mais il s'adresse aux interactions asynchrones et non aux interactions synchrones, comme c'est le cas d'une qualifiant suspendTransaction. Le qualifiant deliverAsyncAt accepte les valeurs call (valeur par défaut) et commit. Si la valeur est call, il indique à l'environnement d'exécution que le message de l'interaction asynchrone doit être validé dans la file d'attente dès que l'appel a été effectué. La valeur commit indique que le message doit être validé dans la file d'attente dans le cadre d'une transaction associée à l'unité de travail en cours.

Qualifiants de réponse asynchrone

Trois qualifiants sont disponibles pour indiquer la qualité de service de la réponse asynchrone. Chacun des qualifiants de réponse asynchrone est spécifié au niveau de la référence. Voici un récapitulatif des qualifiants de réponse asynchrone :

reliability
Le qualifiant reliability permet de spécifier le niveau de qualité de service pour la distribution des messages asynchrones. Le qualifiant reliability accepte les valeurs bestEffort et assured (valeur par défaut).

requestExpiration
Le qualifiant requestExpiration permet de spécifier le délai pendant lequel l'environnement d'exécution doit conserver une demande asynchrone si elle n'a pas encore été distribuée. Une fois que ce délai (en millisecondes) est écoulé, cette demande est supprimée.

responseExpiration
Le qualifiant responseExpiration permet de spécifier le délai pendant lequel l'environnement d'exécution doit conserver une demande asynchrone ou fournir un rappel. La valeur de ce qualifiant est indiquée en millisecondes.

Qualifiants de sécurité

Deux qualifiants sont disponibles pour indiquer la qualité de service relative à la sécurité. Voici un récapitulatif de ces qualifiants :

securityIdentity
Le qualifiant securityIdentity permet de spécifier l'ID sécurité sous lequel l'implémentation du composant de service doit être exécutée lors de la

phase d'exécution. Ce qualifiant doit être placé au niveau de l'implémentation car le composant de service et la valeur indiquée doivent correspondre au nom logique de l'ID sous lequel le composant sera exécuté.

securityPermission
Le qualifiant securityPermission est spécifié au niveau des interfaces et notamment au niveau de l'interface ou de la méthode. La valeur de ce qualifiant indique qu'un appelant de ce service doit posséder le rôle spécifié pour appeler le service.

L'implémentation sous-jacente des qualifiants securityPermission et securityIndentity repose sur les concepts Java EE existants.

Qualifiants de session d'activité

L'ensemble des qualifiants de session d'activité est similaire à celui des qualifiants de transaction présentés plus haut. Le service ActivitySession est une extension du modèle de programmation WebSphere qui peut offrir un autre choix d'unité de travail lorsqu'elle est comparée aux transactions globales. En fait la durée de vie d'un contexte de session d'activité peut être plus longue que celle d'une transaction globale et ce contexte peut même inclure des transactions globales. Voici un récapitulatif des qualifiants de session d'activité :

joinActivitySession
Le qualifiant joinActivitySession est défini au niveau de l'interface et indique si le composant doit rejoindre la session d'activité d'un appelant client. Ce qualifiant accepte deux valeurs : true et false (valeur par défaut). La valeur true indique que l'environnement d'exécution ne doit pas interrompre une session d'activité si elle est présente lorsque le composant est appelé. La valeur false indique qu'une session d'activité doit être suspendue avant l'appel du composant.

activitySession
Le qualifiant activitySession est spécifié au niveau de l'implémentation et permet d'indiquer si une session d'activité doit exister ou non pour exécuter le composant de service auquel elle est associée. Ce qualifiant accepte les valeurs ‘true', ‘false' et 'any‘ (valeur par défaut). Si la valeur est true, le composant sera exécuté dans le cadre d'une session d'activité. Si la valeur est false, le composant ne doit pas être exécuté dans le cadre d'une session d'activité. Cela signifie que le qualifiant joinActivitySession doit également avoir la valeur false pour toutes les interfaces spécifiées pour le composant. Enfin si ce qualifiant a la valeur any, il sera exécuté dans le cadre d'une session d'activité, uniquement s'il en existe une.

suspendActivitySession
Le qualifiant suspendActivitySession est défini au niveau de la référence et permet d'indiquer si un service cible associé à une référence sera appelé ou non dans le cadre de la session d'activité appelante. Si la valeur est true, la session d'activité est interrompue et les méthodes du composant cible ne seront pas exécutées dans le cadre de la session d'activité client. Si la valeur est false (valeur par défaut) la session d'activité n'est pas interrompue et les méthodes du composant cible seront exécutées dans le cadre de la session d'activité client.

Techniques de programmation SCA

Cette section fournit des exemples de technique de programmation SCA (Service Component Architecture).

Concepts associés

«Règles en exécution de la conversion de Java en objets SDO»
Pour une substitution correcte du code généré ou l'identification des éventuelles exceptions d'exécution liées aux conversions de Java en SDO (Service Data Object), il est important de bien comprendre les règles en jeu. La plus grande partie des conversions se font directement, mais il existe des cas complexes où l'environnement d'exécution offre les meilleures possibilités de conversion du code généré.

«Propagation d'en-tête de protocole à partir de liaisons d'exportation non SCA», à la page 174
Le service de contexte est chargé de la propagation du contexte (y compris les en-têtes de protocole comme l'en-tête JMS et le contexte utilisateur comme l'ID de compte) tout au long du chemin d'appel SCA (Service Component Architecture). Le service de contexte offre un ensemble d'API et de paramètres configurables.

Tâches associées

«Remplacement d'une conversion d'objet SDO en Java», à la page 172
Il se peut que la conversion d'un objet SDO (Service Data Object) en objet de type Java effectuée par le système ne réponde pas à vos besoins. Suivez cette procédure pour remplacer l'implémentation par défaut par celle de votre choix.

«Remplacement de l'implémentation d'architecture SCA générée», à la page 173
Il se peut que la conversion de code Java en objet SDO (Service Data Object) effectuée par le système ne réponde pas à vos besoins. Suivez cette procédure pour remplacer l'implémentation d'architecture SCA par défaut par votre propre implémentation.

Règles en exécution de la conversion de Java en objets SDO

Pour une substitution correcte du code généré ou l'identification des éventuelles exceptions d'exécution liées aux conversions de Java en SDO (Service Data Object), il est important de bien comprendre les règles en jeu. La plus grande partie des conversions se font directement, mais il existe des cas complexes où l'environnement d'exécution offre les meilleures possibilités de conversion du code généré.

Types et classes de base

L'environnement d'exécution effectue une conversion directe entre Service Data Objects et types et classes Java de base. Types et classes de base :

- Char or java.lang.Character
- Boolean
- Java.lang.Boolean
- Byte ou java.lang.Byte
- Short ou java.lang.Short
- Int ou java.lang.Integer
- Long ou java.lang.Long
- Float ou java.lang.Float
- Double ou java.lang.Double
- Java.lang.String
- Java.math.BigInteger
- Java.math.BigDecimal
- Java.util.Calendar
- Java.util.Date

- Java.xml.namespace.QName
- Java.net.URI
- Byte[]

Classes et tableaux Java définis par l'utilisateur

Lors de la conversion d'une classe ou d'un tableau (array) Java en SDO, l'environnement d'exécution crée un objet de données possédant un URI généré en inversant le nom de package du type Java et ayant un type égal au nom de la classe Java. Par exemple, si la classe Java `com.ibm.xsd.Customer` est convertie en objet SDO, l'URI est `http://xsd.ibm.com` et le type est Customer. L'environnement d'exécution inspecte ensuite le contenu des membres de la classe Java et affecte les valeurs aux propriétés du SDO.

Lors de la conversion d'un SDO en un type Java, l'environnement d'exécution génère le nom du package en inversant l'URI et le nom du type est égal au type du SDO. Par exemple, l'objet de données de type Customer et dont l'URI est `http://xsd.ibm.com` génère une instance du module Java `com.ibm.xsd.Customer`. L'environnement d'exécution extrait ensuite les valeurs des propriétés du SDO et affecte ces propriétés aux zones de l'instance de la classe Java.

Lorsque la classe Java est une interface définie par l'utilisateur, vous devez substituer le code généré et offrir une classe concrète que l'environnement d'exécution puisse instancier. Si l'environnement d'exécution ne peut créer de classe concrète, une exception se produira.

Java.lang.Object

Si le type Java est java.lang.Object, le type généré est xsd:anyType. Un module peut appeler cette interface avec tout objet SDO. L'environnement d'exception tente d'instancier une classe concrète de la même façon que pour les classes et tableaux (arrays) Java définis par l'utilisateur lorsqu'il ne peut trouver cette classe. Autrement, l'environnement d'exécution passe le SDO à l'interface Java.

Même si la méthode renvoie un objet java.lang.Object, l'environnement d'exécution effectuera la conversion seulement en un SDO, si la méthode renvoie un type concret. L'environnement d'exécution emploie une conversion semblable à celle des classes et tableaux Java définis par l'utilisateur en SDO, tel que décrit dans le paragraphe suivant.

Lors de la conversion d'une classe ou d'un tableau (array) Java en SDO, l'environnement d'exécution crée un objet de données possédant un URI généré en inversant le nom de package du type Java et ayant un type égal au nom de la classe Java. Par exemple, si la classe Java `com.ibm.xsd.Customer` est convertie en objet SDO, l'URI est `http://xsd.ibm.com` et le type est Customer. L'environnement d'exécution inspecte ensuite le contenu des membres de la classe Java et affecte les valeurs aux propriétés du SDO.

Dans un cas ou l'autre, si l'environnement d'exécution est incapable d'accomplir la conversion, une exception se produit.

Classes de conteneur Java.util

Lors de la conversion en une classe de conteneur Java concrète telle que Vector, HashMap, HashSet et autres du même genre, l'environnement d'exécution instanciera la classe de conteneur appropriée. L'environnement d'exécution emploie

une méthode semblable à celle des classes et tableaux Java définis par l'utilisateur pour renseigner la classe de conteneur. Si l'environnement d'exécution ne peut localiser de classe Java concrète, l'environnement d'exécution injectera la classe de conteneur dans le SDO.

Lors de la conversion de classes de conteneur Java concrètes en SDO, l'environnement d'exécution utilise les schémas générés montrés dans la conversion de «Java en XML.»

Interfaces Java.util

Pour certaines interfaces de conteneur du package `java.util`, l'environnement d'exécution instancie les classes concrètes suivantes :

Tableau 17. Conversion de type WSDL en classe Java

Interface	Classes concrètes par défaut
Collection	HashSet
Map	HashMap
List	ArrayList
Set	HashSet

Remplacement d'une conversion d'objet SDO en Java

Il se peut que la conversion d'un objet SDO (Service Data Object) en objet de type Java effectuée par le système ne réponde pas à vos besoins. Suivez cette procédure pour remplacer l'implémentation par défaut par celle de votre choix.

task_prereq

Vérifiez que vous avez généré la conversion de type WSDL vers Java à l'aide de WebSphere Integration Developer ou la commande genMapper.

task_context

Pour remplacer un composant généré qui mappe un type WSDL à un type Java, remplacez le code généré par le code qui répond à vos besoins. Vous pouvez utiliser votre propre mappe si vous avez défini vos propres classes Java. Suivez cette procédure pour effectuer les modifications.

task_procedure

1. Localisez le composant généré. Le nom du composant est *java_class*`Mapper.component`.
2. Editez le composant dans un éditeur de texte.
3. Mettez en commentaires le code généré et insérez votre méthode.

   Ne modifiez pas le nom du fichier qui contient l'implémentation du composant.

task_example

Voici un exemple de composant généré à remplacer :

```
 private Object datatojava_get_customerAcct(DataObject myCustomerID,
    String integer)
{

  // Vous pouvez remplacer ce code par un mappage personnalisé.
  // Mettez en commentaire ce code et écrivez le code personnalisé.
```

```
  // Vous pouvez également changer le type Java transmis au
  // convertisseur que le convertisseur tente de convertir.


  return SDOJavaObjectMediator.data2Java(customerID, integer) ;

}
```

task_postreq

Copiez le composant et les autres fichiers dans le répertoire où se trouve le module conteneur et connectez le composant dans WebSphere Integration Developer ou générez un fichier EAR à l'aide de la commande serviceDeploy.

Remplacement de l'implémentation d'architecture SCA générée

Il se peut que la conversion de code Java en objet SDO (Service Data Object) effectuée par le système ne réponde pas à vos besoins. Suivez cette procédure pour remplacer l'implémentation d'architecture SCA par défaut par votre propre implémentation.

task_prereq

Vérifiez que vous avez généré la conversion de type Java vers WSDL (Web Services Definition Language) en utilisant WebSphere Integration Developer ou la commande genMapper.

task_context

Pour remplacer un composant généré qui mappe un type Java à un type WSDL, remplacez le code généré par le code qui répond à vos besoins. Vous pouvez utiliser votre propre mappe si vous avez défini vos propres classes Java. Suivez cette procédure pour effectuer les modifications.

task_procedure

1. Localisez le composant généré. Le nom du composant est *java_class*`Mapper.component`.
2. Editez le composant dans un éditeur de texte.
3. Mettez en commentaires le code généré et insérez votre méthode.

   Ne modifiez pas le nom du fichier qui contient l'implémentation du composant.

task_example

Voici un exemple de composant généré à remplacer :

```
 private DataObject javatodata_setAccount_output(Object myAccount) {

  // Vous pouvez remplacer ce code par un mappage personnalisé.
  // Mettez en commentaire ce code et écrivez le code personnalisé.

  // Vous pouvez également changer le type Java transmis au
  // convertisseur que le convertisseur tente de convertir.


 return SDOJavaObjectMediator.java2Data(myAccount);


}
```

task_postreq

Copiez le composant et les autres fichiers dans le répertoire où se trouve le module conteneur et connectez le composant dans WebSphere Integration Developer ou générez un fichier EAR à l'aide de la commande serviceDeploy.

Propagation d'en-tête de protocole à partir de liaisons d'exportation non SCA

Le service de contexte est chargé de la propagation du contexte (y compris les en-têtes de protocole comme l'en-tête JMS et le contexte utilisateur comme l'ID de compte) tout au long du chemin d'appel SCA (Service Component Architecture). Le service de contexte offre un ensemble d'API et de paramètres configurables.

Lorsque la propagation du service de contexte est bidirectionnelle, le contexte de réponse écrase systématiquement le contexte en cours. Lorsque vous exécutez un appel d'un composant SCA à un autre, la réponse présente un contexte différent. Un composant de service a un contexte entrant, mais si vous appelez un autre service, celui-ci écrase le contexte sortant d'origine. Le contexte de réponse deviendra alors le nouveau contexte.

Lorsque la propagation du service de contexte est unidirectionnelle, le contexte d'origine reste à l'identique.

Le cycle de vie du service de contexte est associé à un appel. Une demande dispose d'un contexte associé et le cycle de vie de ce context est lié au traitement de cette demande particulière. Lorsque le traitement de cette demande se termine, le cycle de vie de ce contexte s'achève.

Dans le cas d'un processus BPEL (Business Process Execution Language) à court terme, le contexte de réponse écrasera le contexte de demande. Il récupère le contexte de réponse auprès de la première demande et le passera à la demande suivante. Dans le cas d'un d'un processus BPEL au long cours, le contexte de réponse est éliminé par le framework BPEL. Celui-ci stocke le contexte d'origine et utilise ce contexte pour faire d'autres appels sortants.

Les services de contexte possèdent des règles et des tables configurables qui dictent le comportement de la liaison. Pour plus d'informations, voir la documentation API et SPI générées disponible à la section Référence. Lors du développement dans WebSphere Integration Developer, vous pouvez définir le service de contexte sur les propriétés d'importation ou d'exportation. Pour plus de détails, reportez-vous aux informations relatives aux liaisons d'importation et d'exportation dans le centre de documentation de WebSphere Integration Developer.

Programmation des objets métier

Les objets métier sont des conteneurs de données d'application, telles qu'un client ou une facture. Les données sont échangées entre les composants par l'intermédiaire d'objets métier. La structure sous-jacente d'un objet métier est une définition de schéma XML (XSD) et l'accès par programmation aux objets métier est fourni via des interfaces d'objet métier dans WebSphere. Collectivement, ces aspects de l'objet métier, sa représentation structurelle, ses interfaces de programmation, ainsi que son comportement et sa manipulation dans SCA (Service Component Architecture), représentent la structure de l'objet métier, qui fournit des moyens puissants et cohérents de décrire et distribuer les données métier dans votre solution.

Ce document contient des informations sur la programmation des objets métier, avec notamment les descriptions des incidents relatifs à la gestion des constructions de schéma pour certaines fonctions. Pour obtenir des informations sur la procédure de définition d'un objet métier, des instructions de développement d'objets métier et sur l'utilisation des API de programmation d'objets métier, reportez-vous aux articles de la section "Informations connexes".

Concepts associés

«Modèle de programmation»
La section relative au modèle de programmation des objets métier décrit les types de données de base qui sont encapsulées dans la structure des objets métier d'IBM. Pour faciliter la création et la manipulation d'objets métier, la structure des objets métier étend les spécifications d'un objet SDO en fournissant un ensemble de services Java.

«Programmation à l'aide de services d'objet métier», à la page 202
Pour faciliter la création et la manipulation d'objets métier, la structure des objets métier étend les spécifications d'un objet SDO en fournissant un ensemble de services Java. Ces services font partie du package `com.ibm.websphere.bo`.

«Techniques de programmation», à la page 206
Ces techniques illustrent comment programmer efficacement des objets métier à l'aide de la structure des objets métier.

Information associée

Web Services Description Language (WSDL) 1.1

Introduction aux objets SDO (Service Data Objects)

Examen des objets métier dans WebSphere Process Server

Modèle de programmation

La section relative au modèle de programmation des objets métier décrit les types de données de base qui sont encapsulées dans la structure des objets métier d'IBM. Pour faciliter la création et la manipulation d'objets métier, la structure des objets métier étend les spécifications d'un objet SDO en fournissant un ensemble de services Java.

Concepts associés

«Utilisation de l'infrastructure d'objets métier IBM»
La structure des objets métier décrit comment les données de l'environnement d'exécution sont modélisées par les applications, intégrées dans l'environnement d'exécution et représentées en mémoire.

«Modélisation des objets métier», à la page 178
Les données d'objet métier qui circulent dans l'environnement d'exécution de WebSphere Process Server sont modélisées à l'aide de schémas XML. Les schémas XML représentent une alternative aux définitions de type de document (DTD) et permettent d'étendre les fonctionnalités dans les domaines de saisie, héritage et présentation des données. Le schéma XML fournit un formulaire permettant de modéliser les types de données, qui satisfait les normes de l'industrie, qui est largement adopté et qui ne dépend pas de la plateforme et du langage utilisés.

«Modélisation de graphiques métier», à la page 192
Les graphiques métier sont à peut près aux objets métier ce que les graphiques de données SDO sont aux objets de données SDO. Lorsqu'un objet métier de niveau supérieur doit être enrichi pour pouvoir utiliser les services fournis par WebSphere Process Server, il est encapsulé avec un graphique métier. L'encapsuleur de encapsuleur offre un plus dans la mesure où il ajoute des en-têtes de données pour stocker les informations logiquement en mémoire, ou physiquement lors de la sérialisation du graphique métier.

«Modélisation des métadonnées de type objet métier», à la page 197
Les métadonnées de type objet métier peuvent être ajoutées aux définitions d'objets métier pour les optimiser lors de la phase d'exécution. La structure des objets métier permet de concevoir, d'annoter et de convertir des métadonnées de type objet métier.

Utilisation de l'infrastructure d'objets métier IBM

La structure des objets métier décrit comment les données de l'environnement d'exécution sont modélisées par les applications, intégrées dans l'environnement d'exécution et représentées en mémoire.

Le tableau 1 récapitule l'implémentation des types de données de base dans la structure des objets métier.

Tableau 18. Abstractions des données et implémentations correspondantes

Abstraction des données	Implémentation	Description
Données d'instance	Objet métier	Les objets métier sont le mécanisme principal de représentation des entités métier ou de documentation des définitions de messages littéral ; toute une panoplie d'objets peuvent exister, de l'objet de base simple doté de propriétés scalaires à une hiérarchie importante et complexe ou à des graphiques d'objets. Un objet métier est un corollaire direct du concept DataObject de SDO.

Tableau 18. Abstractions des données et implémentations correspondantes (suite)

Abstraction des données	Implémentation	Description
Métadonnées d'instance	Graphique métier	Les graphiques métier sont des encapsuleurs ajoutés autour d'un objet métier simple ou d'une hiérarchie d'objets métier afin de fournir des fonctions supplémentaires, telles que l'intégration d'informations récapitulatives des changements et des événements associés aux objets métier dans le graphique métier. Un graphique métier est un corollaire direct du concept DataGraph de SDO, à la différence près qu'il offre davantage que le simple en-tête de récapitulatif des modifications.
Métadonnées type	Métadonnées d'entreprise Métadonnées de type objet métier	Les métadonnées de type objet métier correspondent aux métadonnées qui peuvent être ajoutées aux définitions d'objets métier pour les optimiser lors de la phase d'exécution. Ces éléments de métadonnées sont ajoutées à la définition de schéma XML de l'objet métier, ainsi que les éléments connus `xs:annotation` et `xs:appinfo`.
Services	Services d'objet métier (API)	Les services d'objets métier sont un ensemble de fonctions fournies en supplément des fonctions de base offertes par les objets SDO de Service Data Objects. Il s'agit par exemple de services de création, de copie, d'égalité et de sérialisation. Ces API se trouvent dans le package `com.ibm.websphere.bo`.

La structure des objets métier de WebSphere Process Server est une extension de la norme SDO. Par conséquent, les objets métier échangés entre les composants WebSphere Process Server correspondent à des instances de la classe `commonj.sdo.DataObject`. Toutefois, la structure des objets métier de WebSphere Process Server ajoute plusieurs services et fonctions qui simplifient et enrichissent la fonctionnalité DataObject de base.

Pour faciliter la création et la manipulation d'objets métier, la structure des objets métier de WebSphere étend les spécifications d'un objet SDO en fournissant un ensemble de services Java. Ces services font partie du package `com.ibm.websphere.bo`.

- **BOFactory** : Service clé qui indique plusieurs méthodes permettant de créer des instances d'objets métier.
- **BOXMLSerializer** : Indique plusieurs méthodes permettant de "développer" un objet métier depuis un flux ou d'écrire le contenu d'un objet métier, au format XML, dans un flux.
- **BOCopy** : Indique plusieurs méthodes permettant de copier des objets métier (sémantique "profonde" et "superficielle").
- **BODataObject** : Vous permet d'accéder aux aspects de l'objet de données d'un objet métier, comme le récapitulatif des modifications, le graphique métier et le récapitulatif de l'événement.
- **BOXMLDocument** : Avant-guichet du service qui vous permet de manipuler l'objet métier comme un document XML.
- **BOChangeSummary** et **BOEventSummary** : Facilite l'accès et la manipulation du récapitulatif des modifications et de la partie récapitulative d'un événement dans un objet métier.
- **BOEquality** : Service qui vous permet de déterminer si deux objets métier contiennent des informations identiques. Il prend en charge l'égalité profonde et superficielle.
- **BOType** et **BOTypeMetaData** : Ces services matérialisent des instances du type `commonj.sdo` et permettent de manipuler les métadonnées associées. Les instances Type peuvent ensuite être utilisées pour créer des objets métier "par type".
- **BOInstanceValidator** : Valide les données composant un objet métier, afin de vérifier s'il est conforme aux éléments XSD.

Modélisation des objets métier

Les données d'objet métier qui circulent dans l'environnement d'exécution de WebSphere Process Server sont modélisées à l'aide de schémas XML. Les schémas XML représentent une alternative aux définitions de type de document (DTD) et permettent d'étendre les fonctionnalités dans les domaines de saisie, héritage et présentation des données. Le schéma XML fournit un formulaire permettant de modéliser les types de données, qui satisfait les normes de l'industrie, qui est largement adopté et qui ne dépend pas de la plateforme et du langage utilisés.

Concepts associés

«Définition d'espace de nom cible»
La plupart des problèmes de communication et des problèmes d'entreprise que XML peut résoudre requièrent une combinaison de plusieurs vocabulaires XML. XML dispose d'un mécanisme permettant d'allouer les noms se qualifiant à des espaces de nom différents, tels que des espaces de nom qui s'appliquent à différents secteurs d'activité. En XML, un identificateur URI fournit un nom unique à associer aux définitions d'élément, d'attribut et de type d'un schéma XML.

«Définition d'objet métier»
WebSphere Process Server offre un mécanisme flexible de définition ou d'importation d'objets métier.

«Définition de propriété d'objet métier», à la page 182
Le schéma XML fournit des types complexes, des types simples et des attributs qui sont utilisés pour générer des objets métier.

«Objets métier hiérarchiques et non hiérarchiques», à la page 188
Les objets métier peuvent être modélisés comme objets hiérarchiques ou non hiérarchiques.

«Caractéristiques des objets métier», à la page 190
Les objets métier possèdent des caractéristiques inhérentes qui améliorent leur utilisation dans la structure des objets métier.

Référence associée

«Artefacts XSD et WSDL pris en charge», à la page 183
Lorsqu'un WSDL ou un schéma est importé dans un projet dans WebSphere Integration Developer, les objets métier affichés à partir du WSDL ou du schéma peuvent ensuite être utilisé pour développer un module. Toutefois, il est important de noter que seuls certains artefacts d'un schéma sont affichés comme *objets métier* (par exemple, les éléments racine/de niveau supérieur et les types complexes nommés). Certains artefacts, tels que les types complexes anonymes imbriqués, *ne sont pas* affichés comme objets métier. Ces restrictions dépendent de quels artefacts sont accessibles dans le schéma XML. Par exemple, si vous importez un schéma qui n'a généré qu'un seul objet métier, il est probable que le reste des éléments correspondait à des types complexes anonymes. Les informations ci-après indiquent quels artefacts XSD et WSDL génèrent des objets métier.

Définition d'espace de nom cible :

La plupart des problèmes de communication et des problèmes d'entreprise que XML peut résoudre requièrent une combinaison de plusieurs vocabulaires XML. XML dispose d'un mécanisme permettant d'allouer les noms se qualifiant à des espaces de nom différents, tels que des espaces de nom qui s'appliquent à différents secteurs d'activité. En XML, un identificateur URI fournit un nom unique à associer aux définitions d'élément, d'attribut et de type d'un schéma XML.

Deux conditions sont requises pour l'espace de nom cible des objets métier :

- Les outils et l'environnement d'exécution des processus métier préfèrent que les espaces de nom cible ressemblent à `http://www.foo.com/xyz` et non à `urn:foo:com:xyz`.
- La structure des objets métier requiert un espace de nom cible pour les objets métier.

Définition d'objet métier :

WebSphere Process Server offre un mécanisme flexible de définition ou d'importation d'objets métier.

Il existe principalement trois formes de schéma XML différentes reconnues par WebSphere Process Server comme définition d'objet métier :

- Une définition de type complexe de niveau supérieur
- Une définition de type complexe anonyme de niveau supérieur
- Un élément de niveau supérieur qui fait référence à un type complexe nommé

Définition de type complexe de niveau supérieur

Voici un exemple de définition de type complexe de niveau supérieur :

```
<complexType name="ProductType">
 <sequence>
  <element name="name" type="string"/>
  <element name="color" type="string" maxOccurs="unbounded"/>
 </sequence>
</complexType>
```

L'importation ou la définition d'objets métier qui sont tous définis à l'aide de définitions de type complexe représente le schéma le plus souple et le plus simple à gérer. Ce modèle offre l'avantage d'une bibliothèque de types qui autorise la réutilisation des types. Cette réutilisation peut être effectuée par l'intermédiaire de trois méthodes différentes.

- Tout d'abord, les nouveaux types peuvent être créés à l'aide du modèle de dérivation des types complexes (modèle d'extension ou de restriction).
- Ensuite, les nouveaux types d'agrégat peuvent être créés à l'aide des types complexes existants et des types simples disponibles comme primitives.
- Enfin, des définitions de document complexes peuvent être créées comme les types complexes d'agrégat.

L'autre implication des objets métier définis comme des types complexes est la suivante : lorsque le type complexe est utilisé par les types de composant `JService` pour transférer des données dans l'environnement d'exécution, pour maintenir la compatibilité avec WS-I, un élément référençant le type complexe nommé doit être créé.

Définition de type complexe anonyme de niveau supérieur

Voici un exemple de définition de type complexe anonyme de niveau supérieur :

```
<element name="Product">
 <complexType>
  <sequence>
   <element name="name" type="string"/>
   <element name="color" type="string" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

Si les objets métier importés correspondent tous à des définitions d'élément anonyme, ils sont prêts à être inclus dans les appels `JService`. Toutefois, ils ne sont pas réutilisables par nature.

Elément de niveau supérieur faisant référence à un type nommé

Voici un exemple d'élément de niveau supérieur faisant référence à un type nommé :

```
<element name="product" type="prod:ProdType"/>
```

Les objets métier qui font référence à des types complexes désignés peuvent être fréquents dans un environnement qui a déjà défini des opérations WSDL nécessitant des définitions d'élément. Dans ce scénario, il est important de prendre en compte la disposition possible des définitions de type complexe et d'élément :

- Les éléments peuvent cohabiter avec leur définition de type complexe dans le même fichier de schéma XML.
- Les éléments peuvent cohabiter avec leur définition de type complexe imbriquées dans un fichier WSDL.
- Les éléments peuvent être définis dans le schéma XML `A.xsd`, tandis que leur définition de type complexe est définie dans le fichier de schéma XML `B.xsd`.
- Les éléments peuvent être imbriqués dans un fichier WSDL, qui fait référence à une définition de type complexe définie dans un fichier de schéma XML.

Exemple

L'exemple illustre tous les mécanismes combinés de définition d'un objet métier.

```
<schema
 targetNamespace="http://www.app.com/Address"
 xmlns:addr="http://www.app.com/Address"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">

 <complexType name="Address">
  <sequence>
   <element name="street1" type="string"/>
   <element name="street2" type="string"/>
   <element name="city" type="string"/>
   <element name="state" type="string"/>
   <element name="zip" type="string"/>
  </sequence>
 </complexType>

 <element name="homeAddress" type="addr:Address"/>
 <element name="workAddress" type="addr:Address"/>
 <element name="otherAddress" type="addr:Address"/>

 <element name="individualContact">
  <complexType>
   <sequence>
    <element name="firstName" type="string"/>
    <element name="lastName" type="string"/>
    <element ref="addr:HomeAddress"/>
    <element ref="addr:WorkAddress"/>
    <element ref="addr:OtherAddress"/>
   </sequence>
  </complexType>
 </element>

 <element name="businessContact">
  <complexType>
   <sequence>
    <element name="name" type="string"/>
    <element ref="addr:WorkAddress"/>
   </sequence>
  </complexType>
 </element>

 <element name="chairmanOfTheBoard">
  <complexType>
   <sequence>
    <element name="startDate" type="date"/>
    <element ref="addr:IndividualContact"/>
```

```
    <element ref="addr:BusinessContact"/>
   </sequence>
  </complexType>
 </element>
</schema>
```

Les instructions suivantes sont celles recommandées pour la définition d'objets métier :

- Les éléments sont définis à l'aide de types nommés ; les types anonymes sont déconseillés.
- Les éléments et les définitions de type complexe ne partagent pas le même schéma XML ou le même fichier WSDL. Cette pratique n'encourage pas la réutilisation des types.
- Les types complexes sont définis dans des fichiers de schéma XML et non des définitions WSDL, ce qui crée une bibliothèque de types telle qu'un concept. Là encore, ce type de définition permet et encourage la réutilisation des types complexes.
- Les définitions d'élément sont générées si nécessaire pour faire référence à une définition de type complexe unique. Par exemple, la définition d'un élément à l'intérieur du WSDL est un modèle encouragé.
- Les définitions d'élément utilisent généralement le même espace de nom cible que leur définition de type complexe.

Définition de propriété d'objet métier :

Le schéma XML fournit des types complexes, des types simples et des attributs qui sont utilisés pour générer des objets métier.

Les définitions de type complexe, les définitions de type complexe anonymes et les éléments référençant des définitions de type complexe sont utilisés pour définir les objets métier externes. La propriété de terme permet de définir les données d'un objet métier. Le terme est dérivé de la propriété de terme SDO (Service Data Object) et est défini par l'interface `commonj.sdo.Property`. Il est synonyme de concept d'un attribut.

Une propriété peut être simple ou complexe. Une propriété simple peut être définie comme attribut de schéma XML ou comme élément de schéma XML avec un type qui est un type simple de schéma XML. Une propriété complexe peut référencer un autre objet métier ou définir une structure complexe dans l'objet métier actuel.

L'intégralité du système des types de schéma XML est prise en charge.

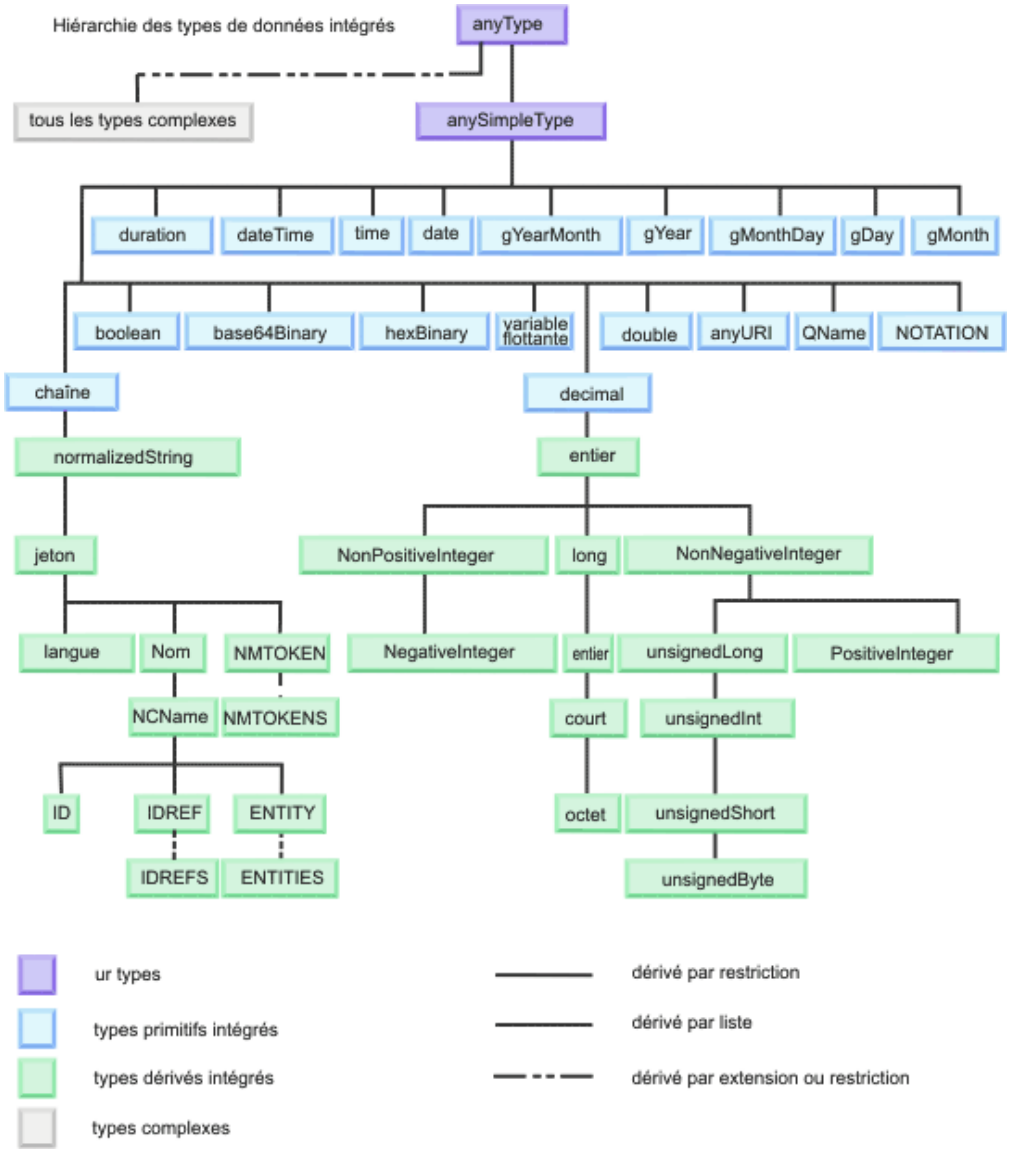


Figure 56. Types simples de schéma XML

Artefacts XSD et WSDL pris en charge :

Lorsqu'un WSDL ou un schéma est importé dans un projet dans WebSphere Integration Developer, les objets métier affichés à partir du WSDL ou du schéma peuvent ensuite être utilisé pour développer un module. Toutefois, il est important de noter que seuls certains artefacts d'un schéma sont affichés comme *objets métier* (par exemple, les éléments racine/de niveau supérieur et les types complexes nommés). Certains artefacts, tels que les types complexes anonymes imbriqués, *ne sont pas* affichés comme objets métier. Ces restrictions dépendent de quels artefacts sont accessibles dans le schéma XML. Par exemple, si vous importez un schéma qui n'a généré qu'un seul objet métier, il est probable que le reste des éléments correspondait à des types complexes anonymes. Les informations ci-après indiquent quels artefacts XSD et WSDL génèrent des objets métier.

Objets métier des définitions XSD importées

Lorsqu'un schéma XML est importé dans un projet, seuls certains artefacts sont affichés comme objets métier. La liste suivante indique quels artefacts sont pris en charge lors des phases de création et d'exécution :

Artefacts XSD générant des objets métier lors de la phase de création :

- Types complexes définis au niveau racine
- Eléments définis au niveau racine avec types complexes anonymes

Ces artefacts génèrent des types simples définis par l'utilisateur lors de la phase de création qui peuvent être référencés par des objets métier :

- Types simples définis au niveau racine
- Eléments définis au niveau racine avec types simples anonymes

Objets métier des fichiers WSDL importés

Lorsqu'une définition WSDL qui inclut un schéma XSD en ligne est importé dans un projet, seuls certains artefacts sont affichés comme objets métier. La liste suivante indique quels artefacts sont pris en charge lors des phases de création et d'exécution :

Artefacts XSD en ligne générant des objets métier lors de la phase de création :

- Types complexes définis au niveau racine
- Eléments définis au niveau de la racine avec des types complexes anonymes ET le nom de l'élément ne contient pas les noms d'opération/de message (car ces éléments peuvent être des éléments doc-lit-wrapped que WebSphere Integration Developer désencapsule automatiquement)

Ces artefacts génèrent des types simples définis par l'utilisateur lors de la phase de création qui peuvent être référencés par des objets métier :

- Types simples définis au niveau racine
- Eléments définis au niveau racine avec types simples anonymes

Objets métier d'exécution des artefacts XSD

Ces artefacts génèrent des objets métier lors de la phase d'exécution :

- Types complexes définis au niveau racine
- Eléments définis au niveau racine avec types complexes anonymes
- Eléments définis au niveau racine qui font référence à un type complexe

Objets métier d'exécution des fichiers WSDL

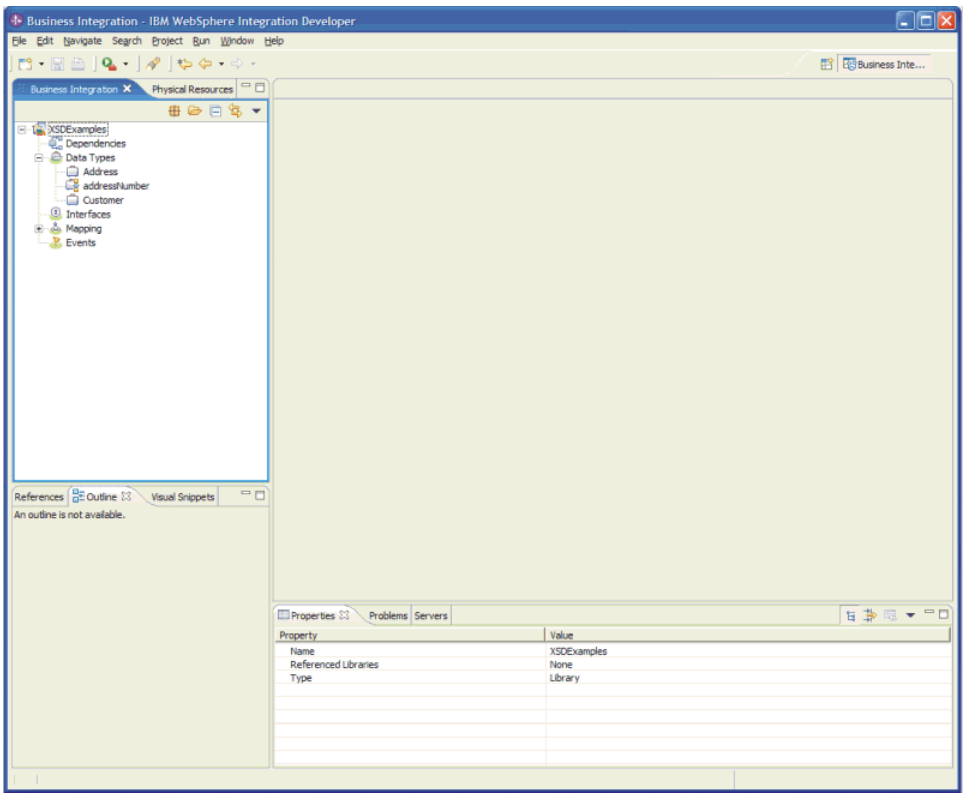
Ces artefacts génèrent des objets métier lors de la phase d'exécution :

- Types complexes définis au niveau racine
- Eléments définis au niveau de la racine avec des types complexes anonymes ET le nom de l'élément ne contient pas les noms d'opération/de message (car ces éléments peuvent être des éléments doc-lit-wrapped que WebSphere Integration Developer désencapsule automatiquement)
- Eléments définis au niveau racine qui font référence à un type complexe

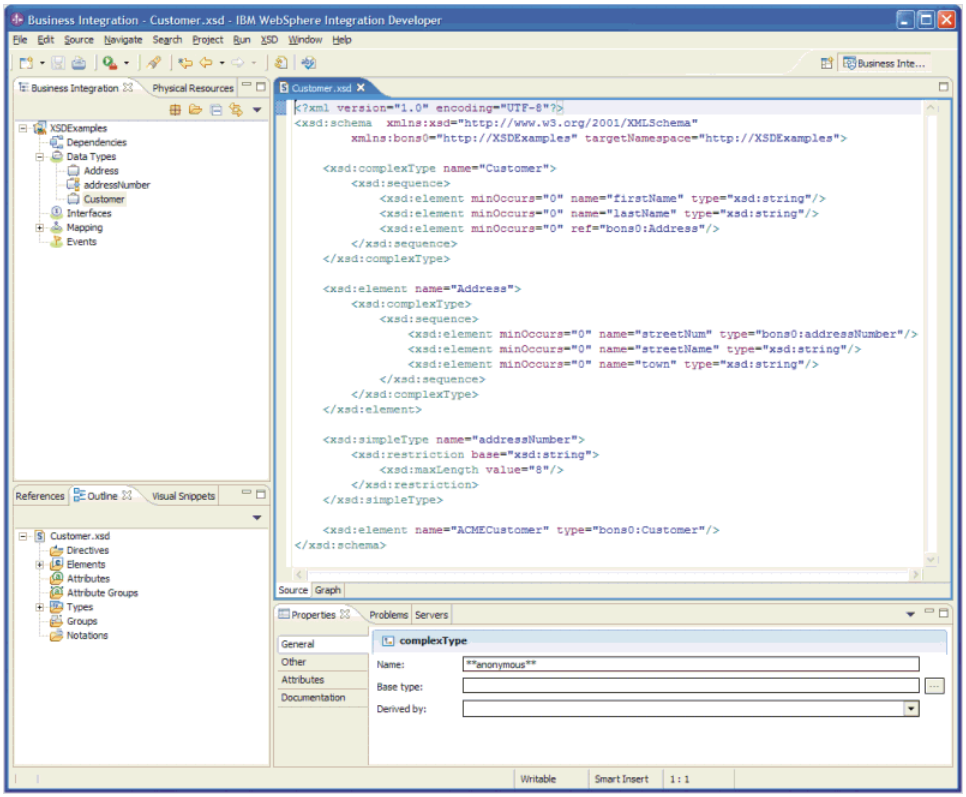
Exemples

1. **Exemple XSD (pris en charge lors de la phase de création)**

   Cet exemple montre un projet (XSDExamples) dans la vue Business Integration, avec les objets métier affichés :



   Cette vue montre le fichier Customer.xsd dans l'éditeur de schéma XSD :



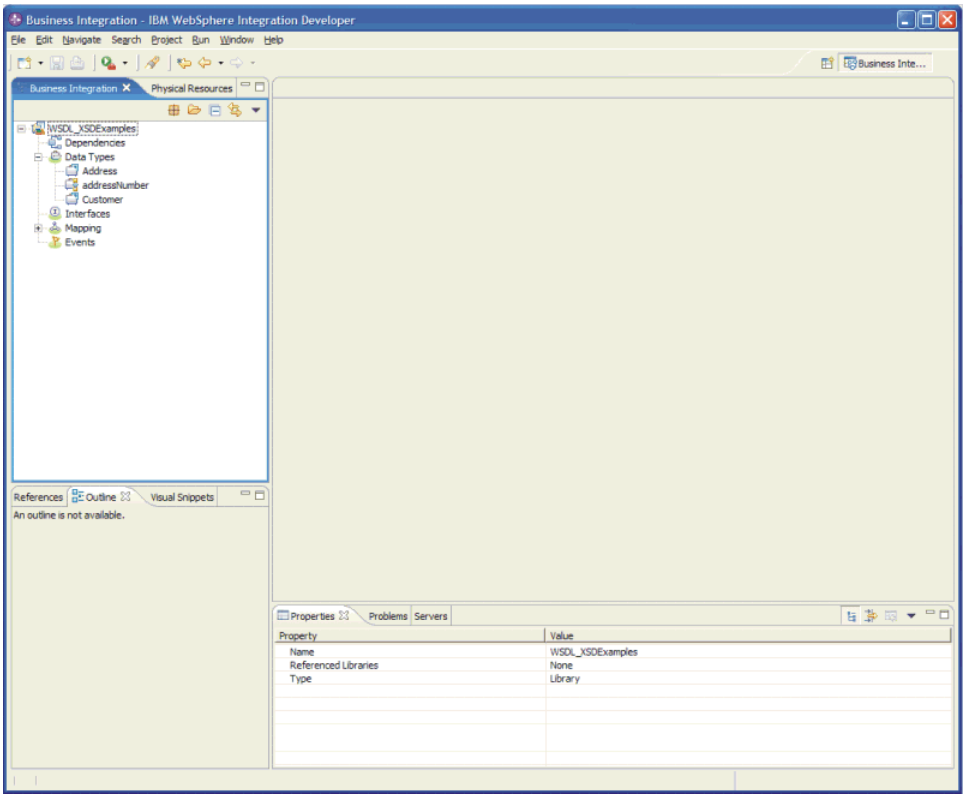
Les exemples suivants illustrent la prise en charge suivante :

Tableau 19. Prise en charge des artefacts XSD

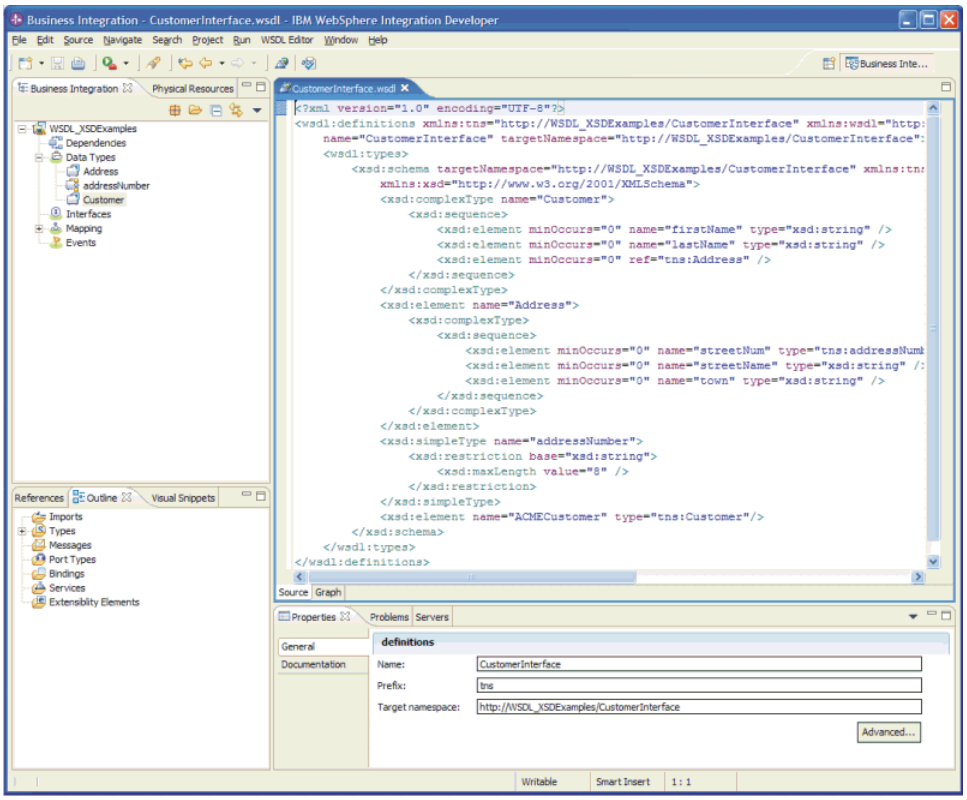
Prise en charge XSD	Artefacts XSD dans l'exemple ci-dessus
Types complexes définis au niveau racine	Client
Eléments définis au niveau racine avec types complexes anonymes	Adresse
Eléments définis au niveau racine avec types simples définis par l'utilisateur	addressNumber

2. **Exemple WSDL (pris en charge lors de la phase de création)**

   Cet exemple montre un projet (WSDL_XSDExamples) dans la vue Business Integration, avec les objets métier affichés :



Cette capture d'écran illustre le fichier CustomerInterface.wsdl ouvert dans l'éditeur WSDL :



Les exemples suivants illustrent la prise en charge suivante :

Tableau 20. Prise en charge des artefacts WSDL

Prise en charge de WSDL	**Artefact XSD en ligne dans l'exemple ci-dessus**
Types complexes définis au niveau racine	Client
Eléments définis au niveau de la racine avec des types complexes anonymes ET le nom de l'élément ne contient *pas* les noms d'opération/de message (car ces éléments peuvent être des éléments doc-lit-wrapped que WebSphere Integration Developer désencapsulera automatiquement)	Adresse
Eléments définis au niveau racine avec types simples définis par l'utilisateur	addressNumber

3. **Exemple d'environnement d'exécution**

   Les exemples suivants illustrent la prise en charge d'environnement d'exécution suivante :

Tableau 21. Prise en charge des artefacts de l'environnement d'exécution

Prise en charge de l'environnement d'exécution	**Artefacts XSD ou XSD en ligne dans les exemples ci-dessus**
Tous les éléments ci-dessus dans les exemples 1 et 2 (excepté addressNumber car les types simples ne sont pas des objets métier)	Voir ci-dessus (exemples 1 et 2)
Eléments définis au niveau racine qui font référence à un type complexe	ACMECustomer (illustré dans les exemples 1 et 2)

Objets métier hiérarchiques et non hiérarchiques :

Les objets métier peuvent être modélisés comme objets hiérarchiques ou non hiérarchiques.

Objet métier non hiérarchique

Un *objet métier non hiérarchique* contient un ou plusieurs attributs simples et une liste d'instructions prises en charge. Un attribut simple représente une valeur, telle qu'une chaîne, un entier ou une date. Tous les attributs simples possèdent une cardinalité unique. Si l'objet métier est spécifique à une application, un objet métier non hiérarchique peut représenter une entité dans une application ou dans une norme technologique.

Objet métier hiérarchique

Les définitions d'*objets métier hiérarchiques* définissent la structure des entités multiple associées, en encapsulant chaque entité individuelle, mais également des aspects de la relation entre les entités. Un objet métier hiérarchique contient au moins un attribut simple, ainsi qu'un ou plusieurs attributs complexes (attribut contenant lui-même un ou plusieurs objets métier, appelés *objets métier enfants*). L'objet métier qui contient l'attribut complexe est appelé *objet métier parent*.

Il existe deux types de relations entre les objets métier parents et enfants :

- Cardinalité unique - Lorsqu'un attribut d'un objet métier parent représente un objet métier enfant unique. Le type de l'attribut correspond au nom de l'objet métier enfant et la cardinalité est "un".
- Cardinalité multiple - Lorsqu'un attribut de l'objet métier parent représente une matrice d'objets métier enfant. Le type de l'attribut correspond au nom de l'objet métier enfant et la cardinalité est *n*.

A leur tour, chacun des objets métier enfant peut contenir des attributs qui contiennent un objet métier enfant ou une matrice d'objet métier enfant, etc. L'objet métier au sommet de la hiérarchie, qui lui-même ne possède pas de parent, est appelé *objet métier de niveau supérieur*. Tout objet métier enfant, indépendamment des objets métier enfant qu'il peut contenir (ou qui peuvent le contenir), est appelé *objet métier individuel*.

Exemple

L'exemple ci-après permet d'illustrer la différence entre un objet métier non hiérarchique et un objet métier hiérarchique. Le diagramme contient un objet métier non hiérarchique, intitulé Product. L'objet métier est représenté en mémoire par le type d'objet SDO `commonj.sdo.DataObject` (sauf s'il a été généré statistiquement). Cet objet métier non hiérarchique possède un ensemble d'attributs modélisés comme types simples de schéma XML, ainsi qu'un attribut modélisé comme une liste de types simples.

Le diagramme illustre également un objet métier Product, avec les objets métier ProductCategory, pour créer un objet métier hiérarchique plus complexe. Cet objet métier possède un objet métier de niveau supérieur (ProductCategory), ainsi qu'un objet métier enfant (Product).

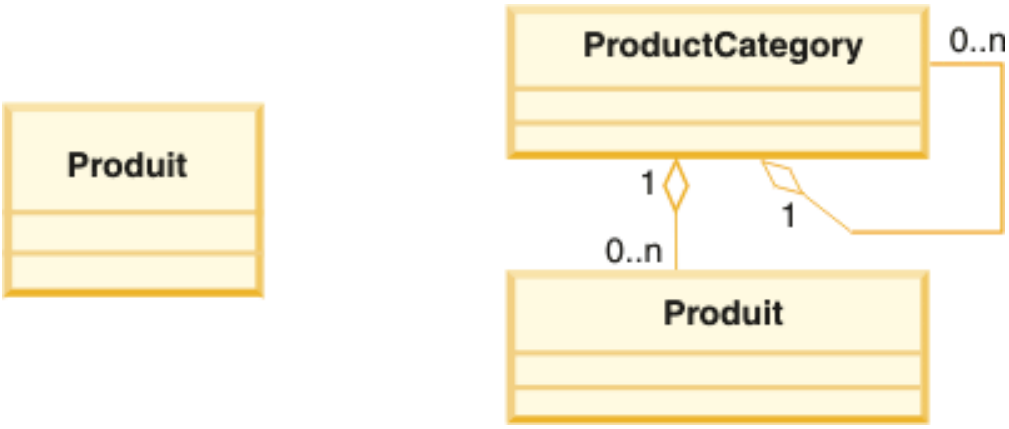


Figure 57. Comparaison des objets métier hiérarchiques et non hiérarchiques

Voici un exemple de définition d'objet métier non hiérarchique pour l'objet métier Product. L'objet métier Product définit deux propriétés, *Name* et *Inventory*, dont le type est défini par les types simples de schéma XML `xs:string` et `xs:int`. En outre, l'objet métier Product démontre également la définition de la propriété de liste *Color*, qui correspond à une liste de types simples `xs:string`.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">
```

```
 <complexType name="Product">
  <sequence>
   <element name="name" type="string"/>
   <element name="inventory" type="int"/>
   <element name="color" type="string" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</schema>
```

Voici un exemple de l'objet métier hiérarchique ProductCategory. La définition définit deux objets métier différents ; ProductCategory et Product. L'objet métier ProductCategory hiérarchique définit la propriété *Name* et une liste d'objets métier de type `Product` ou `ProductCategory`.

```
<schema>
 targetNamespace="http://www.scm.com/ProductCategoryTypes"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema">
 elementFormDefault="qualified">

 <complexType name="ProductCategory">
  <sequence>
   <element name="name" type="string"/>
   <choice>
    <element name=="productCategory"
     type="pc:ProductCategory"
     maxOccurs="unbounded"/>
    <element name=="product"
     type="pc:Product"
     maxOccurs="unbounded"/>
   </choice>
  </sequence>
 </complexType>

 <complexType name="Product">
  <sequence>
   <element name="name" type="string"/>
   <element name=="inventory" type="int"/>
   <element name=="color" type="string" maxOccurs="unbounded"/>
  </sequence>
 </complexType>

</schema>
```

Caractéristiques des objets métier :

Les objets métier possèdent des caractéristiques inhérentes qui améliorent leur utilisation dans la structure des objets métier.

Cardinalité

La cardinalité des propriétés est définie par les facettes `minOccurs` et `maxOccurs` de schéma XML standard pour les types simples et complexes et l'attribut `use` pour les attributs.

Valeurs de propriété par défaut

La capacité à fournir des valeurs par défaut dans le schéma XML pour les attributs et les types simples d'un objet métier est prise en charge par la structure des objets métier. Cette prise en charge est visible lors de la phase de création, lorsque les types de propriété simples d'un objet métier reflètent leurs valeurs par défaut.

Nillable

Un élément peut être défini dans le schéma XML comme "nillable". La structure des objets métier permet aux propriétés de type *nillable* d'accepter la valeur `Null` lors de la phase d'exécution.

Définition de clé

Les informations sur les clés d'objet métier peuvent être utilisées par plusieurs sous-systèmes (par exemple, relation, séquencement et isolement). Toutefois, chacun de ces sous-systèmes peut définir ses propres mécanismes de clé, indépendamment de la définition de clé de l'objet métier. Le langage de modélisation sous-jacent optimisé par les objets métier étant le schéma XML, il existe une prise en charge de premier choix des définitions de clé dans le langage de modélisation. Toutefois, cette prise en charge dans le langage de modélisation n'est pas intégralement prise en charge dans l'environnement d'exécution SDO.

xs:ID, xs:IDREF et xs:IDREFS
: Ces types ont été ajoutés principalement au schéma XML pour fournir un mode de mise à niveau pour les DTD. Chaque type complexe peut comporter 0 ou 1 élément/attribut de type `xs:ID`. Les ID doivent être uniques dans un même document, contrairement à la clé primaire dans une base de données ; par exemple, ils doivent être uniques par rapport à la portée du tableau. Par exemple, un document compatible ne peut pas utiliser la même valeur d'ID pour identifier un produit et une catégorie de produits. Souvent, les éléments contournent cette restriction en ajoutant le nom du type complexe au début de la valeur de la clé. Un attribut typé `IDREF` doit contenir une valeur qui correspond à l'une des valeurs d'ID du document actuel. En outre, le schéma XML fournit une construction qui correspond à un élément dont le type peut contenir une liste de références d'ID : `xs:IDREFs`.

xs:unique, xs:key, xs:keyref
: Le schéma XML a introduit un nouveau style autorisant les définitions de clé et les références de clé. La construction `xs:unique` permet à un utilisateur de déterminer une ou plusieurs zones dans un élément devant être unique dans une portée déterminée de l'élément (représentant le document entier). La construction `xs:key` est une variante de `xs:unique` avec la contrainte supplémentaire que les éléments référencés sont requis. La construction `xs:keyref` permet de déterminer que la valeur d'un élément doit être une clé nommée ou une construction unique.

Les constructions `unique`, `key`, et `keyref` offrent plusieurs avantages par rapport à l'ensemble `ID`, `IDREF` et `IDREFS` et notamment :

- Ils peuvent définir des clés composées.
- Ils peuvent définir des contraintes uniques relatives à une portion du document.

Il n'est pas nécessaire qu'un objet métier possède une clé définie, mais cela est fortement recommandée. Les objets métier qui ne définissent pas de clé peuvent être utilisés par des applications. Ce scénario est un modèle courant dans de nombreux modèles d'utilisation d'applications Java EE, où des beans Java sont transmis entre le servlet et les conteneurs EJB sans la spécification d'une clé. Toutefois, les objets métier qui ne définissent pas de clé ne sont pas capables d'interagir avec les sous-systèmes qui nécessitent une clé. Cette situation limite leur capacité à bénéficier des qualités de service de WebSphere Process Server.

Modélisation de graphiques métier

Les graphiques métier sont à peut près aux objets métier ce que les graphiques de données SDO sont aux objets de données SDO. Lorsqu'un objet métier de niveau supérieur doit être enrichi pour pouvoir utiliser les services fournis par WebSphere Process Server, il est encapsulé avec un graphique métier. L'encapsuleur de encapsuleur offre un plus dans la mesure où il ajoute des en-têtes de données pour stocker les informations logiquement en mémoire, ou physiquement lors de la sérialisation du graphique métier.

Remarque : Si vous migrez une application de WebSphere InterChange Server ou que vous migrez des adaptateurs, vous devez utiliser des graphiques métier.

Concepts associés

«Modèles d'utilisation des graphiques métier»
Deux modèles d'utilisation principaux représentent les fonctions fondamentales fournies par le graphique métier : prise en charge des deltas et image postérieure.

«Définition de modèle de graphique métier», à la page 193
Pour rester non intrusif envers un modèle d'objet métier développé en externe, la fonctionnalité du graphique métier est encapsulée autour de l'objet métier d'origine. Un modèle intitulé graphique métier modèle permet d'encapsuler l'objet métier d'origine avec le schéma de graphique métier enrichi.

«Instance de modèle de graphique métier», à la page 195
Le graphique métier de niveau supérieur est représenté en mémoire à peu près de la même manière qu'un objet métier, par un objet de données SDO 1.0 et plus spécifiquement, la classe `commonj.sdo.DataObject`.

Modèles d'utilisation des graphiques métier :

Deux modèles d'utilisation principaux représentent les fonctions fondamentales fournies par le graphique métier : prise en charge des deltas et image postérieure.

La *prise en charge delta* est la fonctionnalité activée par SDO 1.0, où les modifications apportées à un graphique d'objets métier sont capturées dans un en-tête spécial appelé *récapitulatif des modifications*.

Une *image postérieure* est un graphique métier qui capture l'état actuel des données métier d'un système EIS, généralement à la suite d'une modification de ces données sur le système EIS. Une image postérieure permet la capture et la publication dans l'environnement d'exécution des modifications apportées dans les systèmes EIS.

Pour offrir ces deux concepts fondamentaux, les graphiques métier introduisent et fournissent les concepts suivants :

- Un *graphique métier modèle* est un graphique métier saisi spécifiquement pour un type de graphique d'objet métier qu'il encapsule.
- Un *récapitulatif des modifications* est fourni pour capturer les modifications implicites et les modifications explicites, pour les modèles d'utilisation Delta et Image postérieure.
- La prise en charge du *récapitulatif des modifications explicites* est fournie par les interfaces de programmation de graphique métier qui permettent aux composants BPM, tels que les adaptateurs, les médiateurs, les mappes et les relations, de modifier explicitement l'en-tête du récapitulatif des modifications.
- Le *récapitulatif des événements* est fourni pour prendre en charge la capture des annotations de l'instance sur les données du graphique des objets métier et, spécifiquement, pour conserver les identificateurs des événements d'objet

- La prise en charge des *instructions* est fournie par le graphique métier et permet aux composants de l'environnement d'exécution de décoder le type d'événement pour effectuer une fonction à valeur ajoutée.
- Les *instructions prises en charge* représentent la notion de restriction et d'extension de l'ensemble des instructions admises qui peuvent être spécifiées pour un graphique métier.
- Les *identificateur d'événement d'objet* sont pris en charge par le graphique métier afin que tous les objets d'un graphique puissent être identifiés de manière unique. Cette fonction est requise par certains des composants qui offrent des fonctions à valeur ajoutée.

Définition de modèle de graphique métier :

Pour rester non intrusif envers un modèle d'objet métier développé en externe, la fonctionnalité du graphique métier est encapsulée autour de l'objet métier d'origine. Un modèle intitulé graphique métier modèle permet d'encapsuler l'objet métier d'origine avec le schéma de graphique métier enrichi.

Le *graphique métier modèle* est créé en développant le type complexe de graphique métier fourni par l'environnement d'exécution de la structure des objets métier et en ajoutant un élément qui délègue à l'objet métier d'origine. Le diagramme illustre un modèle UML pour le graphique métier. Le graphique métier est abstrait ; il offre simplement l'ensemble standard des en-têtes ajoutés à l'objet métier de niveau supérieur.

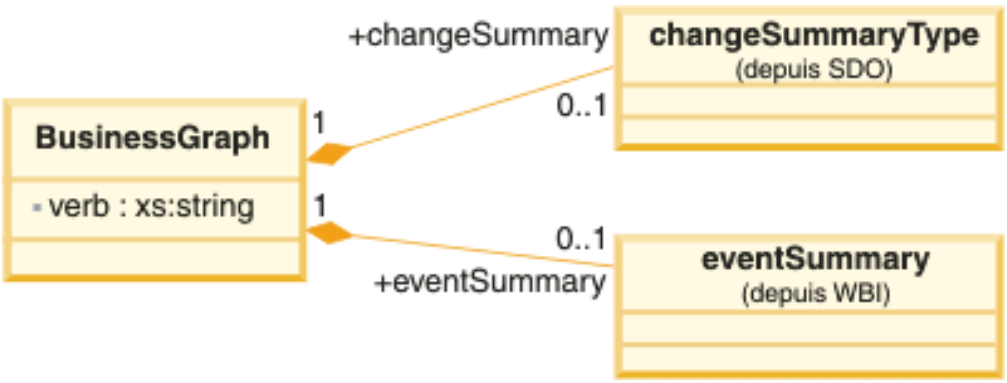


Figure 58. Type complexe de graphique métier

Modèle de schéma XML pour le type complexe de schéma XML de graphique métier :

```
<schema
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:sdo="commonj.sdo"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="commonj.sdo" schemaLocation="DataGraph.xsd"/>

 <complexType name="BusinessGraph" abstract="true">
  <sequence>
   <element name="changeSummary" type="sdo:ChangeSummaryType"
    minOccurs="0" maxOccurs="1"/>
   <element name="eventSummary" type="bo:EventSummary"
```

```
    minOccurs="0" maxOccurs="1"/>
   <element name="property" type="bo:ValuesType"
    minOccurs="0"/>
  </sequence>
  <anyAttribute namespace="##other" processContents="lax"/>
 </complexType>

 <complexType name="EventSummary">
  <sequence>
   <any namespace="##any" processContents="lax"
    minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
 </complexType>

 <complexType name="ValuesType">
  <complexContent>
   <extension base="ecore:EClass"/>
  </complexContent>
 </complexType>

 <attribute name="name" type="string"/>
</schema>
```

Les graphiques métier ne représentent qu'un concept global car ils servent à ajouter un ensemble d'en-têtes à un objet métier de niveau supérieur existant et ne peuvent pas être modélisés dans un modèle de conception récursif, comme les objets métier. Les graphiques métier peuvent être appliqués à tout objet métier, mais lors de leur application, cet objet métier devient un objet métier de niveau supérieur.

Voici un exemple de graphique métier qui encapsule un objet métier intitulé ProductCategory ; ProductCategory est un objet métier hiérarchique qui contient un objet métier enfant intitulé Product.

```
<schema
 targetNamespace="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pcbg="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="http://www.ibm.com/xmlns/prod/websphere/bo/6.0.0"
  schemaLocation="BusinessGraph.xsd"/>

 <import namespace="http://www.scm.com/ProductCategoryTypes"
  schemaLocation="ProductCategoryTypes.xsd"/>

 <complexType name="ProductCategoryBG">
  <complexContent>
   <extension base="bo:BusinessGraph">
    <sequence>
     <element name="verb" minOccurs="0" maxOccurs="1"/>
     <element name="productCategory"
      type="pc:ProductCategory"
      minOccurs="0" maxOccurs="1"/>
    </sequence>
   </extension>
  <complexContent>
 <complexType>
</schema>
```

Le modèle recommandé pour un graphique métier modélisé généré est le suivant :

- Les graphiques métier modélisés sont définis à l'aide d'un type complexe nommé qui étend le schéma `bo:BusinessGraph` par restriction (ce modèle fournit des restrictions sur la valeur d'instruction autorisable).
- Le nom du graphique métier modélisé correspond au nom de l'objet métier de niveau supérieur auquel la chaîne "BG" est ajoutée.
- L'espace de nom cible du graphique métier modélisé est composé de l'espace de nom cible de l'objet métier encapsulé, suivi d'un caractère "/", nommé par le type complexe du graphique métier modélisé.
- La définition de type complexe du graphique métier modélisé est placée dans son propre fichier de schéma XML dont le nom correspond au nom du type complexe.

Instance de modèle de graphique métier :

Le graphique métier de niveau supérieur est représenté en mémoire à peu près de la même manière qu'un objet métier, par un objet de données SDO 1.0 et plus spécifiquement, la classe `commonj.sdo.DataObject`.

Le graphique métier n'est pas composé uniquement de l'objet de confinement du graphique métier. Il contient également deux en-têtes plus l'objet métier de niveau supérieur. Aucun des en-têtes n'est représenté en mémoire comme objet de données et n'active de mécanisme d'accès à l'API DataObject.

En-tête du récapitulatif des modifications

Les graphiques métier permettent de rechercher implicitement les modifications apportées à l'objet métier dans le graphique métier lorsque ce dernier est transmis entre plusieurs processus métier différents. A mesure que chaque processus modifie le graphique métier, un journal des modifications est généré en mémoire. Une fois que le graphique métier est sérialisé, le journal des modifications est enregistré dans un format qui permet au prochain processus de voir les types de modification apportés au graphique métier. Cette technique permet à l'adaptateur et aux services DMS de mettre à jour efficacement leurs magasins de données de persistance en optimisant les données qui les intéressent.

En outre, le récapitulatif des modifications est également utilisé lorsqu'un adaptateur génère un événement d'image postérieure pour décrire les données mises à jour dans un système EIS. En particulier, la capacité du récapitulatif des modifications à pouvoir annoter les modifications apportées aux objets et aux propriétés est utilisée dans le modèle d'utilisation Image postérieure (ainsi que l'instruction du graphique métier).

Utilisation implicite du récapitulatif des modifications

Lorsque des applications modifient l'objet métier, elles peuvent activer la consignation des modifications pour que les événements de modification soient automatiquement consignés dans le récapitulatif des modifications. Le récapitulatif des modifications est inclus dans le graphique métier au niveau des homologues de l'objet métier de niveau supérieur. Les événements de modification sont définis à deux niveaux : pour les objets métier et pour les propriétés d'objet métier. Les objets métier possèdent des types de modification create, update et delete, tandis que les propriétés possèdent des types de modification set et unset. Les événements de modification ne sont recherchés que pour les modifications apportées à la portion "objet métier" du graphique métier. Une modification dans le récapitulatif des modifications n'entraîne pas de mise à jour implicite du

récapitulatif des modifications du graphique métier.

Modification explicite du récapitulatif des modifications

Plusieurs modèles d'utilisation des composants WebSphere Process Server requièrent la possibilité d'écrire explicitement dans l'en-tête du récapitulatif des modifications. Par exemple, un adaptateur qui génère un événement EIS crée explicitement les types de modification d'objet et éventuellement les types de modification de propriétés. Une mappe d'objet métier spécifique à l'application/d'objet métier général (ASBO/GBO) transforme le récapitulatif des modifications d'un graphique métier en un autre, créant ainsi une nouvelle version du récapitulatif des modifications dans un graphique métier en sortie. Cette fonctionnalité est offerte par la structure des objets métier.

En-tête du récapitulatif des événements

Le récapitulatif des événements fournit le mécanisme **ObjectEventID**, qui permet d'identifier de manière unique une instance d'un objet qui apparaît dans l'environnement d'exécution. Ces informations sont transportées dans le récapitulatif des événements, où l'identificateur unique est associé à un objet de données spécifié dans la hiérarchie des objets métier du graphique métier.

Les informations sur les événements peuvent également être transportées dans le récapitulatif des événements. Ces informations se présentent sous la forme d'une chaîne qui peut être utilisée pour ajouter des métadonnées supplémentaires associées à chaque objet de la hiérarchie des objets métier du graphique métier. Un modèle d'utilisation potentiel pour les informations sur les événements consiste à marquer les objets métier contenus avec une instruction autre que les instructions Create, Update et Delete standard prises en charge par le récapitulatif des modifications.

En-tête de l'instruction

Si l'instruction est définie dans le graphique métier, la portion données de l'objet métier du graphique métier transporte un fichier d'image postérieure de système EIS. Si l'instruction contient une valeur, trois cas sont possibles en matière de granularité de l'événement d'image postérieure :

- Le récapitulatif des modifications est vide. Cette situation survient lorsqu'un système EIS connaît le type de mise à jour d'un fichier, mais qu'ils ne dispose pas de données spécifiques sur les objets du graphique qui ont été créés, mis à jour ou supprimés. Par conséquent, une mappe, une relation, un adaptateur ou une médiation en aval doit utiliser les informations du graphique métier, ainsi que des données supplémentaires pour déterminer la véritable mise à jour à effectuer.
- Le récapitulatif des modifications contient des annotations d'événement de modification au niveau des objets. Ce cas se produit lorsque le système EIS reconnaît ce qui est arrivé à chaque objet du graphique métier, mais qu'il n'a pas la granularité requise pour déterminer si les propriétés spécifiques des objets ont été mises à jour.
- Le récapitulatif des modifications contient des annotations d'événement de modification au niveau des objets et des annotations get/set au niveau des propriétés. Il s'agit du cas le plus granulaire et tous les adaptateurs s'efforcent d'obtenir ce niveau d'image postérieure si leur système EIS leur permet d'accéder aux données appropriées. Une image postérieure intégralement spécifiée permet la gestion des événements de modification des propriétés

implicites au niveau des propriétés. Par conséquent, il est bien plus simple pour les graphiques métier des images postérieures d'interagir avec les graphiques métier delta déconnectés.

Modélisation des métadonnées de type objet métier

Les métadonnées de type objet métier peuvent être ajoutées aux définitions d'objets métier pour les optimiser lors de la phase d'exécution. La structure des objets métier permet de concevoir, d'annoter et de convertir des métadonnées de type objet métier.

La structure des objets métier offre un mécanisme qui :

- Permet de mélanger les métadonnées dans un objet métier de manière cohérente et relativement non intrusive
- Active une règles prescriptive permettant aux développeurs de définir des structures d'annotation complexes
- Active une règles prescriptive permettant aux développeurs d'objets métier et aux déployeurs d'annoter un objet métier avec des métadonnées d'instance qui respectent les structures d'annotation complexes prédéfinies
- Active un ensemble d'API permettant de transformer les annotations lors de la phase d'exécution en une structure d'objets de données facile à utiliser

Cette procédure implique au moins trois rôles différents :

- Le premier rôle est celui du concepteur des métadonnées de type objet métier. Ce rôle conçoit la structure des métadonnées. Par exemple, la structure des objets métier joue ce rôle et définit quelques caractéristiques de métadonnées pour annoter u objet métier. Les adaptateurs tels que PeopleSoft, Siebel et SAP doivent jouer le rôle de concepteur de métadonnées pour annoter l'objet métier avec des informations spécifiques à l'application.
- Le deuxième rôle est celui du concepteur ou du déployeur d'objets métier. Ce rôle utilise la structure des métadonnées de type objet métier et la règle définie par la structure des métadonnées de type objet métier pour annoter les définitions d'objet métier avec des métadonnées.
- Si l'objet métier est annoté correctement, le troisième rôle peut utiliser les API des métadonnées d'objet métier pour valider et inspecter les métadonnées lors de la phase d'exécution et les transformer en une structure de graphiques d'objet de données navigable et utile.

En outre, la structure des objets métier offre les fonctions de métadonnées suivantes :

- Définitions de propriété de clé externe et de clé principale composée
- Annotations de métadonnées d'instruction de niveau supérieur prises en charge

Pour plus de détails, reportez-vous à ces autres rubriques.

Concepts associés

«Représentation des métadonnées de type objet métier»
La structure des objets métier définit un mécanisme à partir duquel les métadonnées de type objet métier peuvent être mélangées dans un schéma importé descendant ou développé en externe.

Tâches associées

«Conception des métadonnées de type objet métier», à la page 199
Vous pouvez concevoir une structure de métadonnées pour héberger les métadonnées des objets métier.

«Annotation d'une définition d'objet métier», à la page 199
Vous pouvez utiliser la syntaxe prise en charge par la structure des objets métier pour annoter une définition d'objet métier.

«Conversion d'annotations en objets de données», à la page 200
L'infrastructure des objets métier permet de convertir des annotations en une structure d'objets de données utilisable.

Représentation des métadonnées de type objet métier :

La structure des objets métier définit un mécanisme à partir duquel les métadonnées de type objet métier peuvent être mélangées dans un schéma importé descendant ou développé en externe.

Le mécanisme permettant de mélanger les métadonnées de type objet métier sont les annotations de schéma XML et la structure `appInfo`. Cette règle, bien qu'elle nécessite de modifier la définition de schéma d'origine, le fait d'une manière qui permet de consulter facilement les annotations et `appinfo`, de basculer entre elles ou de les supprimer intégralement. Ce schéma d'identification est réalisé en utilisant l'attribut source sur la balise `xs:appinfo` dont la valeur correspond à l'espace de nom cible qui définit les métadonnées de type objet métier.

Supposons par exemple, que le schéma ci-après ait été importé dans l'environnement d'exécution. Le schéma décrit l'objet métier, intitulé Product, qui inclut les annotations client.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:p="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified

 <complexType name="Product">
  <annotation>
   <appInfo>
    <SCMEditor value="Bottom" type="Anchor"/>
   </appInfo>
   <documentation>
    Describes the SCM Product
   </documentation>
  </annotation>

  <sequence>
   <element name="id" type="ID"/>
   <element name="description" type="string" default="DefaultDescription"/>
   <element name="sku" type="p:Sku"/>
  </sequence>
 </complexType>

 <simpleType name="Sku">
  <restriction base="string">
```

```
   <pattern value="\d{3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>
</schema>
```

Conception des métadonnées de type objet métier :

Vous pouvez concevoir une structure de métadonnées pour héberger les métadonnées des objets métier.

task_context

Pour concevoir la structure des métadonnées, procédez comme suit :

task_procedure

1. Créez un fichier de schéma XML avec un espace de nom cible valide. Cette étape est utilisée lorsque les métadonnées de l'instance sont ajoutées à la définition d'objet métier.
2. Définissez un type complexe nommé pour définir chaque portion de métadonnées distincte qui peut être ajoutée à un objet métier. La définition du type complexe permet de définir la structure de l'objet de données (`DataObject`) saisi dynamiquement qui permet de lire les métadonnées de l'instance. Le nom du type complexe est utilisé lorsque les métadonnées de l'instance sont ajoutées à la définition d'objet métier.

task_example

Cet exemple illustre une portion des métadonnées de type objet métier développées pour un adaptateur PeopleSoft.

```
<schema>
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/adapter/psft/
  PSFTBODefinitionASI/7.0.0"
 xmlns:psft="http://www.ibm.com/xmlns/prod/websphere/adapter/psft/
  PSFTBODefinitionASI/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <complexType name="PSFTBODefinitionASI">
  <sequence>
   <element name="hostname" type="string"/>
   <element name="ipaddress" type="string"/>
  </sequence>
 </complexType>
</schema>
```

Annotation d'une définition d'objet métier :

Vous pouvez utiliser la syntaxe prise en charge par la structure des objets métier pour annoter une définition d'objet métier.

task_context

La structure des objets métier ne tente pas de définir le mécanisme par l'intermédiaire duquel les métadonnées d'instance sont associées à la définition d'objet métier. Toutefois, elle ne définit pas la syntaxe des métadonnées d'instance.

Pour annoter la définition d'objet métier, suivez les instructions ci-après.

task_procedure

1. Si la partie de la définition d'objet métier à laquelle les métadonnées d'instance doivent être associées ne contient pas déjà une annotation, ajoutez-en une. Si elle en contient déjà une, utilisez-la.
2. Créez une balise `xs:appinfo` distincte dans la balise `xs:annotation` et spécifiez comme attribut source l'espace de nom défini par les métadonnées de type objet métier ajoutées.
3. Dans la balise `xs:appinfo`, définissez un QName, à l'aide d'un préfixe d'espace de nom cible, suivi du nom de la définition de type complexe. Ajoutez l'attribut QName défini avec `xmlns:` suivi du préfixe d'espace de nom cible précédemment utilisé. Spécifiez comme valeur de cet attribut QName l'espace de nom cible des métadonnées de type objet métier utilisées.
4. Ajoutez la structure et les données d'instance mandatées par la définition des métadonnées de type objet métier. Fermez toutes les balises comme il se doit.

task_example

Une fois que l'objet est importé, il est identifié comme nécessitant un ensemble d'annotations PeopleSoft. Cet exemple illustre la même définition d'objet métier à laquelle les métadonnées PeopleSoft ont été ajoutées.

```
<schema>
 targetNamespace="http://www.scm.com/ProductTypes"
 xmlns:p="http://www.scm.com/ProductTypes"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified

 <complexType name="Product">
  <annotation>
   <appinfo source="http://www.ibm.com/xmlns/prod/websphere/Adapter/PSFT">
    <psft:PSFTMetadata
     xmlns:psft="http://www.ibm.com/xmlns/prod/websphere/Adapter/PSFT">
     <hostname>mumbai</hostname>
     <ipaddress>9.29.1.1</ipaddress>
    <psft:PSFTMetadata>
   </appInfo>
   <appInfo>
    <SCMEditor value="Bottom" type="Anchor"/>
   </appInfo>
   <documentation>
    Describes the SCM Product
   </documentation>
  </annotation>

  <sequence>
   <element name="id" type="ID"/>
   <element name="description" type="string" default="DefaultDescription"/>
   <element name="sku" type="p:Sku"/>
  </sequence>
 </complexType>

 <simpleType name="Sku">
  <restriction base="string">
   <pattern value="\d{3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>
</schema>
```

Conversion d'annotations en objets de données :

L'infrastructure des objets métier permet de convertir des annotations en une structure d'objets de données utilisable.

task_context

Les annotations associées à une définition d'objet métier peuvent être lues lors de la phase d'exécution, à l'aide de l'ensemble d'API spécifique à l'implémentation SDO. Le problème qui se pose est lié au fait que ces API renvoient un objet binaire de grande taille (BLOB). Toutefois, l'infrastructure d'objets métier fournit un utilitaire qui, si les modèles d'annotation sont respectés, lit l'objet BLOB, le valide et le transforme en structure d'objet de données utilisable.

Pour convertir une annotation en objet de données, procédez comme suit.

task_procedure

1. Procurez-vous une annotation.
2. Utilisez BOTypeMetadata pour convertir cette annotation en un objet de données SDO. L'implémentation BOTypeMetadata est disponible comme singleton par le biais de l'instance BOTypeMetadata.INSTANCE.

task_example

L'exemple suivant indique comme utiliser les API pour obtenir une annotation et utiliser ensuite l'APIBOTypeMetadata pour convertir cette dernière en un objet de données SDO. Les métadonnées sont définies dans `BOTypeMetadata.xsd`.

```
<schema>
 targetNamespace="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <complexType name="VerbInfo">
  <sequence>
   <element name="verbInfo" type="string"/>
  </sequence>
 </complexType>
</schema>
```

L'infrastructure d'objets métier offre comme fonctionnalité l'ajout d'instructions prises en charge supplémentaires et des métadonnées associées afin de bénéficier des avantages offerts par ces dernières au cours de l'exécution. Cette capacité est prise en charge via l'utilisation d'instructions et de métadonnées VerbInfo visant à annoter les instructions avec des métadonnées supplémentaires. L'exemple suivant indique l'emplacement où les métadonnées VerbInfo doivent être ajoutées pour chacune des valeurs d'instruction (Verb) possibles.

```
<schema
 targetNamespace="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pcbg="http://www.scm.com/ProductCategoryTypes/ProductCategoryBG"
 xmlns:pc="http://www.scm.com/ProductCategoryTypes"
 xmlns:sdo="commonj.sdo"
 xmlns:bo="http://www.ibm.com/xmlns/prod/websphere/bo/7.0.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">

 <import namespace="http://www.ibm.com/xmlns/prod/websphere/bo/7.0.0"
  schemaLocation="BusinessGraph.xsd"/>

 <import namespace="commonj.sdo"
```

```
 schemaLocation="DataGraph.xsd"/>

<import namespace="http://www.scm.com/ProductCategoryTypes"
 schemaLocation="ProductCategoryTypes.xsd"/>

<complexType name="ProductCategoryBG">
 <complexContent>
  <extension base="bo:BusinessGraph">
   <sequence>

    <element name="verb"
     minOccurs="0" maxOccurs="1">
     <simpleType>
      <restriction base="string">
       <enumeration value="Create">
        <annotation>
         <appinfo source="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
          <botm:VerbInfo
           xmlns:botm="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
           <verbInfo>Metadata relating to Create</verbInfo>
          </botm:VerbInfo>
         </appinfo>
        </annotation>
       </enumeration>
       <enumeration value="Retrieve">
        <annotation>
         <appinfo source="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
          <botm:VerbInfo
           xmlns:botm="http://www.ibm.com/xmlns/prod/websphere/botm/7.0.0">
           <verbInfo>Metadata relating to Retrieve</verbInfo>
          </botm:VerbInfo>
         </appinfo>
        </annotation>
       </enumeration>
      </restriction>
     </simpleType>
    </element>

    <element name="productCategory" type="pc:ProductCategory"
     minOccurs="0" maxOccurs="1"/>

   </sequence>
  </extension>
 </complexContent>
</complexType>

<element name="productCategoryBG" type="pcbg:ProductCategoryBG"/>

</schema>
```

Programmation à l'aide de services d'objet métier

Pour faciliter la création et la manipulation d'objets métier, la structure des objets métier étend les spécifications d'un objet SDO en fournissant un ensemble de services Java. Ces services font partie du package `com.ibm.websphere.bo`.

Voici une brève description des services de l'objet métier.

Tableau 22. Services de l'objet métier

Maintenance	Description
BOChangeSummary	Améliore l'interface du récapitulatif des modifications SDO pour gérer l'en-tête du récapitulatif des modifications du graphique métier.

Tableau 22. Services de l'objet métier (suite)

Maintenance	Description
BOCopy	Facilite la copie d'un graphique d'objets métier ou d'un graphique métier qui contient un graphique d'objets métier.
BODataObject	Améliore l'interface des objets de données SDO.
BOEquality	Permet de déterminer si deux graphiques métier ou objets métier sont équivalents.
BOEventSummary	Offre l'interface de gestion du contenu de l'en-tête de récapitulatif des événements du graphique métier.
BOFactory	Permet de créer un graphique métier ou un objet métier.
BOType	Fournit un mécanisme permettant d'obtenir le type SDO d'un graphique métier ou d'un objet métier qui reflète ce que `Class.forName()` fournit pour les noms de classe Java.
BOTypeMetadata	Permet de transformer un objet binaire de grande taille (BLOB) qui respecte le modèle des métadonnées de type objet métier en un ensemble d'objets de données SDO (et effectue la transformation de réserve).
BOXMLDocument/BOXMLSerializer	Fournit le mécanisme permettant de créer et de représenter un document XML en mémoire et de sérialiser et désérialiser un document XML.
BOInstanceValidator	Offre un utilitaire permettant de valider une instance d'objet métier par rapport à sa définition XSD. Les objets métier peuvent se présenter sous plusieurs formes. Il peut s'agir d'objets métier simples ou encapsulés par le modèle de graphique métier enrichi. Dans certains scénarios d'intégration métier, les objets métier se trouvent dans la section supprimée du récapitulatif des modifications. Ces objets métier gèrent les logiques métier en aval. La précision des objets métier doit être assurée dans tous les cas. Deux styles sont pris en charge pour BOInstanceValidator : • Validation par programme explicite : Un service système est fourni pour valider les objets métier via un ensemble d'API de programmation. • Validation d'interface implicite : Cette validation est activée/désactivée via WebSphere Integration Developer sur les interfaces cible via un qualificateur d'interface SCA.

Concepts associés

«Validation de document XML»
Il est possible de valider les documents XML et les objets métier à l'aide du service de validation.

Validation de document XML

Il est possible de valider les documents XML et les objets métier à l'aide du service de validation.

Par ailleurs, d'autres services requièrent un certain nombre de standards minimum sinon une exception d'exécution est émise. `BOXMLSerializer` est l'un de ces services.

Vous pouvez utiliser `BOXMLSerializer` pour valider des documents XML avant qu'ils ne soient traités par une demande de service. `BOXMLSerializer` valide la structure des documents XML pour déterminer s'il comporte l'un des types d'erreur suivants :

- Documents XML non valides, comme ceux pour lesquels il manque certaines balises d'éléments.
- Documents XML syntaxiquement incorrects, comme ceux pour lesquels il manque des balises fermantes.
- Documents contenant des erreurs d'analyse syntaxique, comme les erreurs dans les déclarations d'entité.

Lorsque `BOXMLSerializer` identifie une erreur, une exception est émise avec une analyse détaillée de l'incident.

Vous pouvez valider l'importation et/ou l'exportation des documents XML pour les services suivants :

- HTTP
- Services Web JAXRPC
- Services Web JAX-WS
- Services JMS
- Services MQ

Pour les services HTTP, JAXRPC et JAX-WS, `BOXMLSerializer` génère les exceptions de la manière suivante :

- Importations –
  1. Le composant SCA appelle le service.
  2. Le service appelle l'URL d'une destination.
  3. L'URL cible répond avec une exception XML non valide.
  4. Le service échoue avec une exception d'exécution et un message.
- Exportations –
  1. Le client de service appelle l'exportation de service.
  2. Le client de service envoie un XML non valide
  3. L'exportation échoue pour le service et génère une exception et un message.

Pour les services de messagerie JMS et MQ, les exceptions sont générées de la manière suivante :

- Importations –
  1. L'importation appelle le service JMS ou MQ.
  2. Le service renvoie une réponse.

3. Le service renvoie une exception XML non valide.
4. L'importation échoue et génère un message.

- Exportations –
  1. Le client MQ ou JMS appelle une exportation.
  2. Le client envoie un XML non valide.
  3. L'exportation échoue et génère une exception et un message.

Vous pouvez afficher les journaux pour tout message généré par une exception de validation XML. Les exemples ci-dessous représentent des messages générés par du code XML incorrect qui a été validé par `BOXMLSerializer`

- Importation JAXWS

```
javax.xml.ws.WebServiceException: org.apache.axiom.om.OMException:
 javax.xml.stream.XMLStreamException: Element type "TestResponse" must be
 followed by either attribute specifications, ">" or "/>".

javax.xml.ws.WebServiceException: org.apache.axiom.soap.SOAPProcessingException:
 First Element must contain the local name, Envelope
```

- Importation JAXRPC

```
[9/11/08 15:16:27:417 CDT] 0000003e ExceptionUtil E
 CNTR0020E: EJB threw an unexpected (non-declared)
  exception during invocation of method
  "transactionNotSupportedActivitySessionNotSupported" on bean
  "BeanId(WSXMLValidationApp#WSXMLValidationEJB.jar#Module, null)".
  Exception data: WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: org.xml.sax.SAXParseException: Element type "TestResponse"
   must be followed by either
  attribute specifications, ">" or "/>". Message being parsed:
    <?xml version="1.0"?><TestResponse
  xmlns="http://WSXMLValidation"<firstName>Bob</firstName>
    <lastName>Smith</lastName></TestResponse>
 faultActor: null
 faultDetail:
[9/11/08 15:16:35:135 CDT] 0000003f ExceptionUtil E   CNTR0020E: EJB threw an
  unexpected (non-declared) exception during invocation of method
  "transactionNotSupportedActivitySessionNotSupported" on bean
  "BeanId(WSXMLValidationApp#WSXMLValidationEJB.jar#Module, null)".
 Exception data: WebServicesFault
 faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.generalException
 faultString: org.xml.sax.SAXException: WSWS3066E: Error: Expected 'envelope'
   but found  TestResponse
  Message being parsed: <?xml version="1.0"?><TestResponse
   xmlns="http://WSXMLValidation">
  <firstName>Bob</firstName><middleName>John</middleName>
   <lastName>Smith</lastName>
   </TestResponse>
 faultActor: null
 faultDetail:
```

- Exportation JAXRPC/JAXWS

```
[9/11/08 15:35:13:401 CDT] 00000064 WebServicesSe E
  com.ibm.ws.webservices.engine.transport.http.WebServicesServlet
  getSoapAction WSWS3112E:
   Error: Generating WebServicesFault due to missing SOAPAction.
                    WebServicesFault
 faultCode: Client.NoSOAPAction
 faultString: WSWS3147E: Error: no SOAPAction header!
 faultActor: null
 faultDetail:
```

Pour plus d'informations sur les services de validation, voir l'interface de `BOInstanceValidator` dans la documentation sur l'interface SPI et l'interface de programme d'application générées de la section Référence.

Techniques de programmation

Ces techniques illustrent comment programmer efficacement des objets métier à l'aide de la structure des objets métier.

Concepts associés

«Tableaux dans les objets métier»
Vous pouvez définir des tableaux pour un élément dans un objet métier afin que cet élément puisse contenir plus d'une instance de données.

«Création d'objets métier imbriqués», à la page 208
La fonction `setWithCreate` permet de créer des objets métier imbriqués au sein d'un objet métier parent.

«Différenciation d'éléments portant le même nom», à la page 212
Vous devez donner des noms uniques aux éléments et attributs d'objet de données.

«Différenciation de propriétés portant le même nom», à la page 215
Lorsque plusieurs XSD avec le même espace de nom définissent des types portant le même nom, un type incorrect peut être accidentellement référencé.

«Résolution de noms de propriétés contenant des points», à la page 216
Les noms des propriétés dans un XSD peuvent contenir un point (".") comme un des nombreux caractères valides, alors que, dans un SDO, ils sont également utilisés pour montrer l'indexation dans une propriété à cardinalité multiple. Dans certaines situations, ceci peut entraîner des problèmes de résolution.

«Prise en charge des objets métier null», à la page 218
Ce scénario implique un système externe communiquant avec WebSphere Process Server via du code XML encapsulé dans un message SOAP. Si l'élément imbriqué est marqué "nillible" et que `xsi:nil="true"`, l'objet de données résultant qui est créé dans WebSphere Process Server est null.

«Utilisation de l'objet de séquence pour définir l'ordre des données», à la page 218
Certains XSD sont définis de telle sorte que l'ordre des données dans le XML a une importance significative.

«Utilisation de tout type de données», à la page 222
Cette section fournit des techniques de programmation permettant d'utiliser tout type de données.

Tableaux dans les objets métier

Vous pouvez définir des tableaux pour un élément dans un objet métier afin que cet élément puisse contenir plus d'une instance de données.

Vous pouvez utiliser une Liste pour créer un tableau pour un seul élément nommé dans un objet métier. Vous pourrez ainsi utiliser cet élément pour contenir des instances multiples de données. Par exemple, vous pouvez utiliser un tableau pour stocker plusieurs numéros de téléphone dans un élément nommé `telephone` et défini en tant que chaîne dans l'encapsuleur d'objet métier. Vous pouvez également définir la taille du tableau en précisant le nombre d'instances de données. Pour cela, vous utiliserez la valeur`maxOccurs`. L'exemple de code suivant montre comment créer un tel tableau comportant trois instances de données pour cet élément :

```
<xsd:element name="telephone" type="xsd:string" maxOccurs="3"/>
```

Cela va créer un index pour l'élément telephone qui peut contenir jusqu'à trois instances de données. Vous pouvez également utiliser la valeur minOccurs si vous envisagez d'avoir un élément dans le tableau.

Le tableau créé se compose de deux éléments :

- son contenu
- le tableau lui-même.

Pour créer ce tableau, cependant, vous devez effectuer une opération intermédiaire consistant à définir un encapsuleur. Celui-ci remplace en effet la propriété de l'élément par un objet tableau. Dans l'exemple ci-dessus, vous pouvez créer un objet ArrayOfTelephone pour définir l'élément telephone en tant que tableau. L'exemple de code suivant indique comment accomplir cette tâche :

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="Customer">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="ArrayOfTelephone" type="ArrayOfTelephone"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="ArrayOfTelephone">
      <xsd:sequence maxOccurs="3">
        <xsd:element name="telephone" type="xsd:string" nillable="true"/>
      </xsd:sequence>
    </xsd:complexType>
 </xsd:schema>
```

L'élément telephone apparaît maintenant en tant qu'enfant de l'objet encapsuleur ArrayOfTelephone.

Vous remarquerez que dans l'exemple ci-dessus, l'élément telephone comprend la propriété nillable. Vous pouvez définir cette valeur sur true si vous voulez que certains éléments dans l'indice de tableau ne contiennent aucune donnée. L'exemple de code suivant indique comment les données d'un tableau peuvent être représentées :

```
<Customer>
   <name>Bob</name>
   <ArrayOfTelephone>
     <telephone>111-1111</telephone>
     <telephone xsi:nil="true"/>
     <telephone>333-3333</telephone>
   </ArrayOfTelephone>
 </Customer>
```

Dans ce cas, le premier et le troisième éléments dans l'indice de tableau de l'élément telephone contiennent des données contrairement au deuxième. Si vous n'aviez pas utilisé la propriété nillable pour l'élément telephone, vous auriez alors les deux premiers éléments qui contiennent des données.

Vous pouvez utiliser les API de séquence Service Data Object (SDO) dansWebSphere Process Server comme alternative au traitement des séquences dans les tableaux d'objet métier. L'exemple de code suivant permet de créer un tableau pour l'élément telephone avec des données identiques à celles indiquées plus haut :

```
DataObject customer = ...
 customer.setString("name", "Bob");

 DataObject tele_array = customer.createDataObject("ArrayOfTelephone");
 Sequence seq = tele_array.getSequence();  // The array is sequenced
 seq.add("telephone", "111-1111");
 seq.add("telephone", null);
 seq.add("telephone", "333-3333");
```

Vous pouvez renvoyer les données d'un indice de tableau d'élément donné en utilisant un code semblable à l'exemple ci-dessous :

```
String tele3 = tele_array.get("telephone[3]");  // tele3 = "333-3333"
```

Dans cet exemple, la chaîne `tele3` va renvoyer les données "333-3333".

Vous pouvez entrer les éléments de données du tableau dans l'index en utilisant une largeur fixe ou des données délimitées placées dans une file d'attente de messages JMS ou MQ. Vous pouvez également accomplir cette tâche en utilisant un fichier texte à plat contenant les données correctement formatées.

Création d'objets métier imbriqués

La fonction `setWithCreate` permet de créer des objets métier imbriqués au sein d'un objet métier parent.

Vous pouvez créer des objets métier imbriqués à partir d'un objet métier parent sans écrire de code détaillant les objets enfant intermédiaires. Par exemple, vous pouvez créer un objet métier imbriqué deux niveaux sous l'objet parent sans avoir à définir un objet métier dépendant entre les deux, c'est-à-dire un niveau sous l'objet parent. La fonction `setWithCreate` permet d'accomplir cette tâche pour :

- une seule instance
- plusieurs instances
- une valeur de caractère générique
- un groupe de modèles

Les rubriques suivantes expliquent comment procéder.

Tâches associées

«Instance unique d'un objet métier imbriqué»
La fonction `setWithCreate` permet de créer une instance unique d'un objet métier imbriqué.

«Création de plusieurs instances d'objets métier imbriqués», à la page 209
La fonction `setWithCreate` permet de créer des instances multiples d'un objet métier imbriqué.

«Utilisation d'un objet métier imbriqué défini par un caractère générique», à la page 211
Vous pouvez spécifier le type `xsd:any` dans un objet parent pour indiquer un objet enfant, mais uniquement si cet objet enfant existe déjà.

«Utilisation des objets métier dans les groupes de modèles», à la page 211
Vous créez les modèles de chemin de groupe de modèles lorsque vous utilisez des objets métier imbriqués faisant partie d'un groupe de modèles.

Instance unique d'un objet métier imbriqué :

La fonction `setWithCreate` permet de créer une instance unique d'un objet métier imbriqué.

task_prereq

L'exemple suivant montre comment vous devriez normalement créer du code pour un objet intermédiaire (enfant) à partir d'un objet de niveau plus élevé (parent) afin de créer un objet de troisième niveau (grand-enfant). Le fichier XSD aurait la forme suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

task_context

Si vous utilisiez la méthode traditionnelle "descendante" pour définir les données d'objet métier, il vous faudrait traiter le code suivant précisant les objets enfant et grand-enfant avant de définir les données dans l'objet grand-enfant :

```
DataObject parent = ...
DataObject child = parent.createDataObject("child");
DataObject grandchild = child.createDataObject("grandChild");
grandchild.setString("name", "Bob");
```

Il existe une méthode plus efficace qui consiste à utiliser la fonction `setWithCreate`. Celle-ci permet en effet de définir simultanément l'objet grand-enfant et ses données, sans avoir à préciser l'objet enfant intermédiaire. L'exemple de code suivant indique comment accomplir cette tâche :

```
DataObject parent = ...
parent.setString("child/grandchild/name", "Bob");
```

task_results

L'objet métier de niveau inférieur est définir sans avoir à définir l'objet métier de niveau intermédiaire. Une exception est émise toutefois si le chemin est incorrect.

Création de plusieurs instances d'objets métier imbriqués :

La fonction `setWithCreate` permet de créer des instances multiples d'un objet métier imbriqué.

task_prereq

L'exemple suivant représente un fichier XSD contenant des objets imbriqués se trouvant un niveau (enfant) et deux niveaux (petit-enfant) sous l'objet métier supérieur (parent) :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="Child" maxOccurs="5"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Child">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="grandChild" type="GrandChild"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="GrandChild">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Vous remarquerez que l'objet parent peut avoir jusqu'à cinq objets enfant, comme l'indique la valeur `maxOccurs`.

task_context

Vous pouvez créer une liste avec une règle plus rigoureuse ne permettant pas que des séquences soient absentes d'un tableau. Vous pouvez utiliser la méthode `setWithGet` et, en même temps, préciser les données qui apparaîtront dans un élément de l'index de liste particulier :

```
DataObject parent = ...
parent.setString("child[3]/grandchild/name", "Bob");
```

Dans cet exemple, vous obtenez un tableau de taille trois, mais les valeurs des éléments de l'index de liste `child[1]` et `child[2]` ne sont pas définies. Vous voudrez peut-être leur attribuer la valeur null ou la valeur d'une donnée associée. Dans le scénario ci-dessus, une exception sera émise car la valeur des deux premiers éléments du tableau n'est pas définie.

Vous pouvez remédier à cette situation en définissant ces valeurs dans l'index de la liste. Si l'élément de l'index fait référence à un élément existant du tableau et que la valeur de cet élément n'est pas null (c'est-à-dire qu'il contient des données), celui-ci sera utilisé. Si sa valeur est null, il sera créé puis utilisé. Si l'index de la liste est plus grand que la taille de celle-ci, une nouvelle valeur sera créée et ajoutée. L'exemple suivant illustre le fonctionnement dans une liste de taille deux, où l'élément `child[1]` est désigné comme null et l'élément `child[2]` contient des données :

```
DataObject parent = ...
// child[1] = null
// child[2] = existing Child
// Ce code fonctionne car l'élément child[1] est null et sera créé.
```

```
parent.setString("child[1]/grandchild/name", "Bob");

// Ce code fonctionne car l'élément child[2] existe et sera utilisé.
parent.setString("child[2]/grandchild/name", "Dan");

// Ce code fonctionne car la liste enfant est de taille 2 et l'ajout
// d'un élément de liste supplémentaire va accroître la taille de la liste.
parent.setString("child[3]/grandchild/name", "Sam");
```

task_results

Vous avez remplacé les valeurs des deux éléments existants et ajouté un troisième à l'index de la liste. Néanmoins, si vous ajoutez un autre élément qui n'est pas de taille quatre, ou qui est plus grand que la taille précisée dans `maxOccurs`, une exception sera émise. La règle plus rigoureuse de cette méthode est démontrée dans l'exemple suivant.

Remarque : Le code qui suit est ajouté au code utilisé ci-dessus :

```
// Ce code entraîne une exception car la liste est de taille 3
// et vous n'avez pas créé d'élément pour augmenter la taille à 4.
parent.setString("child[5]/grandchild/name", "Billy");
```

Utilisation d'un objet métier imbriqué défini par un caractère générique :

Vous pouvez spécifier le type `xsd:any` dans un objet parent pour indiquer un objet enfant, mais uniquement si cet objet enfant existe déjà.

task_context

La fonction `setWithCreate` utilisée pour définir des objets métier imbriqués pour une seule ou plusieurs instances ne fonctionne pas si vous utilisez la valeur générique `xsd:any` dans l'objet de données de service. L'exemple suivant illustre cette situation :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Parent">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="child" type="xsd:anyType"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

task_results

Une exception sera émise si l'objet données enfant n'existe pas.

Utilisation des objets métier dans les groupes de modèles :

Vous créez les modèles de chemin de groupe de modèles lorsque vous utilisez des objets métier imbriqués faisant partie d'un groupe de modèles.

task_context

Les groupes de modèles utilisent la balise `xsd:choice` que vous pouvez utiliser pour créer des objets métier à partir d'un objet métier parent. Toutefois, la structure

des objets métier, peut entraîner des conflits de dénomination qui peuvent alors générer une exception. L'exemple suivant illustre comment les conflits d'attribution de noms peuvent se produire :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
        <xsd:element name="child1" type="Child"/>
        <xsd:element name="child2" type="Child"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Remarque : Il peut y avoir plusieurs instances des éléments "child1" et "child2"

Utilisez les modèles de chemin Service Data Object (SDO) pour les groupes de modèles pour résoudre ces conflits.

task_results

Vous obtiendrez des tableaux qui utilisent le modèle de chemin SDO utilisé pour traiter les groupes de modèles, comme indiqué dans l'exemple de code suivant :

```
set("child1/grandchild/name", "Bob");

set("child11/grandchild/name", "Joe");
```

Différenciation d'éléments portant le même nom

Vous devez donner des noms uniques aux éléments et attributs d'objet de données.

Dans l'infrastructure SDO, les éléments et les attributs sont créés en tant que propriétés. Dans les exemples de code suivants, les XSD créent des types comportant une propriété nommée foo :

```
<xsd:complexType name="ElementFoo">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AttributeFoo">
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>
```

Dans ces cas-là, vous pouvez accéder à la propriété en utilisant le langage XML Path (XPath). Cependant, les types de schéma valides peuvent comporter un attribut et un élément qui portent le même nom, comme dans l'exemple suivant :

```
<xsd:complexType name="DuplicateNames">
  <xsd:sequence>
    <xsd:element name="foo" type="xsd:string" default="elem_value"/>
  </xsd:sequence>
  <xsd:attribute name="foo" type="xsd:string" default="attr_value"/>
</xsd:complexType>
```

Dans XPath, vous devez pouvoir différencier des éléments portant le même nom des attributs. Pour cela, on ajoute au début des noms le symbole (@). Le fragment suivant montre comment accéder à un élément et un attribut portant le même nom :

```
1 DataObject duplicateNames = ...

2 // Affiche "elem_value"
3 System.out.println(duplicateNames.get("foo"));

4 // Affiche "attr_value"
5 System.out.println(duplicateNames.get("@foo"));
```

Utilisez ce schéma de désignation pour toutes les méthodes prenant une valeur de chaîne dans un XPath SDO.

Concepts associés

«Prise en charge de groupes de modèles (tous, choix, séquence et références de groupes)»
La spécification SDO nécessite que les groupes de modèles (tous, choix, séquence et références de groupes) soient développés et ne décrit pas les types ni les propriétés.

Prise en charge de groupes de modèles (tous, choix, séquence et références de groupes) :

La spécification SDO nécessite que les groupes de modèles (tous, choix, séquence et références de groupes) soient développés et ne décrit pas les types ni les propriétés.

Pratiquement, cela signifie que toutes les structures qui se trouvent dans les mêmes structures sont "mises à plat". Cette "mise à plat" met tous les enfants de ces structures au même niveau, ce qui peut entraîner des problèmes de noms dupliqués dans un SDO dont la structure est dérivée des données mises à plat. Lorsqu'un XSD ne met pas à plat les groupes, les noms dupliqués contenus par des parents différents restent séparés.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://MultipleGroup">
  <xsd:complexType name="MultipleGroup">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
      <xsd:element name="separator" type="xsd:string"/>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Les occurrences multiples de option1 et option2 se trouvant dans des blocs de choix distincts, et comportant même un élément de séparation entre eux, le XSD et le XML les distingue sans problème. Mais lorsque le SDO met à plat ces groupes, toutes les propriétés d'option sont maintenant sous le même conteneur de groupe multiple.

Même sans noms dupliqués, la mise à plat de ces groupes entraîne un problème d'ordre sémantique. Par exemple, pour le XSD suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://SimpleChoice">
  <xsd:complexType name="SimpleChoice">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="option1" type="xsd:string"/>
        <xsd:element name="option2" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Demander à l'utilisateur de renommer les noms dupliqués ou d'ajouter des annotations spéciales aux XSD n'est pas possible dans beaucoup de cas, comme les schémas de normes ou industriels, car l'utilisateur ne contrôle pas les XSD avec lesquels il travaille.

Pour que toutes les propriétés soient cohérentes, les objets métier incluent une méthode pour accéder à chaque occurrence individuelle des propriétés portant le même nom via la balise XPath. Selon la convention de dénomination de la structure des objets métier, le chiffre non utilisé suivant est ajouté à tous les noms dupliqués trouvés ; par exemple, le XSD suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://TieredGroup">
  <xsd:complexType name="TieredGroup">
    <xsd:sequence>
      <xsd:choice minOccurs="0">
        <xsd:sequence>
          <xsd:element name="low" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:choice minOccurs="0">
            <xsd:element name="width" minOccurs="0"
      maxOccurs="1" type="xsd:string"/>
            <xsd:element name="high" minOccurs="0"
      maxOccurs="1" type="xsd:string"/>
          </xsd:choice>
        </xsd:sequence>
        <xsd:element name="high" minOccurs="1"
    maxOccurs="1" type="xsd:string"/>
          <xsd:sequence>
            <xsd:element name="width" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:element name="high" minOccurs="0"
     maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element name="center" minOccurs="1"
     maxOccurs="1" type="xsd:string"/>
          <xsd:element name="width" minOccurs="0"
     maxOccurs="1" type="xsd:string"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Le XSD précédent produit le modèle d'objet de données suivant :

```
DataObject - TieredGroup
 Property[0] - low - string
 Property[1] - width - string
 Property[2] - high - string
 Property[3] - high1 - string
 Property[4] - width1 - string
 Property[5] - high2 - string
 Property[6] - center - string
 Property[7] - width2 - string
```

Où **width**, **width1** et **width2** sont les noms des propriétés nommées "width" en commençant par la première dans le XSD et ainsi de suite, et de même pour **high**, **high1**, **high2**.

Les nouveaux noms des propriétés sont les noms utilisés pour référence et XPath et n'affectent pas le contenu sérialisé. Les noms "vrais" de chacune de ces propriétés apparaissant dans le XML sérialisé sont les valeurs données dans le XSD. Ainsi, pour l'instance XML :

```
<?xml version="1.0" encoding="UTF-8"?>
<p:TieredGroup xsi:type="p:TieredGroup"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://TieredGroup">
  <width>foo</width>
  <high>bar</high>
</p:TieredGroup>
```

Pour accéder à ces propriétés, vous devez utiliser le code suivant :

```
 DataObject tieredGroup = ...

 // Affiche "foo"
 System.out.println(tieredGroup.get("width1"));

 // Affiche "bar"
 System.out.println(tieredGroup.get("high2"));
```

Différenciation de propriétés portant le même nom

Lorsque plusieurs XSD avec le même espace de nom définissent des types portant le même nom, un type incorrect peut être accidentellement référencé.

Address1.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="city" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

Address2.xsd:
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Address">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="state" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Les objets métier ne prennent pas en charge les noms dupliqués pour des structures XSD globales (telles que complexType, simpleType, element, attribute, etc.) par le biais des API BOFactory.create(). Il est cependant possible de créer ces

structures globales dupliquées comme enfants d'autres structures si les API correctes sont utilisées, comme indiqué dans les exemples suivants

Customer1.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Customer1"
 targetNamespace="http://Customer1">
  <xsd:import schemaLocation="./Address1.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Customer2.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Customer2"
 targetNamespace="http://Customer2">
  <xsd:import schemaLocation="./Address2.xsd"/>
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element minOccurs="0" name="address" type="Address"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Lorsque les champs "Customer address" sont renseignés et que l'API BOFactory.create() est appelée pour créer l'adresse, les types d'objets métier enfants qui en résultent peuvent être définis d'une manière incorrecte. Pour éviter cela, vous pouvez appeler l'API createDataObject("address") sur l'objet de données "Customer". Un enfant de type correct sera ainsi créé, car les objets métier correspondront à l'emplacement de schéma de l'importation.

```
DataObject customer1 = ...

// Manière incorrecte de créer un enfant "Address"
// Un type d'adresse Address1.xsd ou Address2.xsd risquerait d'être créé
DataObject incorrect = boFactory.create("", "Address");
customer1.set("address", incorrect);

// Manière correcte de créer un enfant "Address"
// Le type d'adresse Address1.xsd sera ainsi forcément créé
customer1.createDataObject("address");
```

Résolution de noms de propriétés contenant des points

Les noms des propriétés dans un XSD peuvent contenir un point (".") comme un des nombreux caractères valides, alors que, dans un SDO, ils sont également utilisés pour montrer l'indexation dans une propriété à cardinalité multiple. Dans certaines situations, ceci peut entraîner des problèmes de résolution.

Les noms des propriétés dans les objets de données de service (SDO) sont basés sur les noms des éléments et de l'attribut à partir desquels ils sont générés dans le XSD. Les objets métier traiteront le caractère "." correctement, avec une exception : si un XSD comporte une propriété à cardinalité unique dont le nom est "<name>.<#>" et une propriété à cardinalité multiple dont le nom est "<name>".

Une balise XPath telle que "foo.0" ne sera pas résolue correctement s'il y a une propriété à cardinalité unique nommée "foo.0" et une propriété à cardinalité

multiple appelée "foo". Dans ce cas, la propriété à cardinalité unique portant le nom "foo.0" est celle qui sera résolue. Bien que cela ne risque de se produire que rarement, vous pouvez l'éviter entièrement si vous utilisez la syntaxe "foo[1]" pour accéder à leur propriété à cardinalité multiple. Les SDO ne prendront pas en charge la syntaxe "." pour l'indexation, et vous devez donc utiliser "[]" pour l'indexation.

Concepts associés

«Sérialisation et désérialisation d'unions portant xsi:type»
Dans le XSD, une union est un moyen de fusionner les espaces lexicaux de plusieurs types de données simples connus comme membres.

Sérialisation et désérialisation d'unions portant xsi:type :

Dans le XSD, une union est un moyen de fusionner les espaces lexicaux de plusieurs types de données simples connus comme membres.

L'exemple de XSD suivant montre une union comportant les membres d'un nombre entier et d'une date.

```
<xsd:simpleType name="integerOrDate">
 <xsd:union memberTypes="xsd:integer xsd:date"/>
</xsd:simpleType>
```

Cette saisie multiple peut entraîner une confusion lors de la désérialisation et de la manipulation des données.

Les objets métier prennent en charge les SDO utilisant xsi:type pour la sérialisation et suivront le même algorithme pour déterminer le type lors d'une désérialisation si le xsi:type n'est pas présent dans les données XML.

Ainsi, pour garantir que les données (le nombre "42" dans cet exemple) seront désérialisées comme un nombre entier, vous pouvez utiliser le xsi:type spécifié dans le XML d'entrée. Vous pouvez également ordonner la liste des membres de l'union dans le XSD de telle sorte que le nombre entier soit avant la chaîne. L'exemple suivant montre comment les deux méthodes sont mises en oeuvre :

```
<integerOrString xsi:type="xsd:integer">42</integerOrString>

<xsd:simpleType name="integerOrString">
 <xsd:union memberTypes="xsd:integer xsd:string"/>
</xsd:simpleType>
```

De même, si l'utilisateur souhaitait que les données soient désérialisées en tant que chaîne, l'une ou l'autre des modifications suivantes entraînerait le comportement suivant :

```
<integerOrString xsi:type="xsd:string">42</integerOrString>

<xsd:simpleType name="integerOrString">
 <xsd:union memberTypes="xsd:string xsd:integer"/>
</xsd:simpleType>
```

Remarque : si un type de chaîne est le premier membre de l'union, aucune de ses informations n'est jamais perdue. Il peut également contenir toutes les données qui seront toujours choisies par l'algorithme no xsi:type. Si vous souhaitez utiliser un autre type qu'une chaîne, vous devez soit utiliser xsi:type dans le XML soit réorganiser les types de membre dans le XSD pour donner aux autres membres la possibilité d'accepter les données.

Prise en charge des objets métier null

Ce scénario implique un système externe communiquant avec WebSphere Process Server via du code XML encapsulé dans un message SOAP. Si l'élément imbriqué est marqué "nillible" et que `xsi:nil="true"`, l'objet de données résultant qui est créé dans WebSphere Process Server est null.

Voici un exemple illustrant un message XML comportant un élément marqué "nillible".

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
 <soap:Body>
  <p:Employee xmlns:p="http://www.mycompany.com" xmlns:xsi="http://www.w3.org"
   xsi:nil="true"/>
 </soap:Body>
</soap:Envelope>
```

Où `Employee` est défini comme suit :

```
<element name="Employee" nillable="true">
...
</element>
```

L'objet métier généré et envoyé par l'exportation est null dans ce cas. Par exemple, si des opérations sont appelées sur des composants en aval, l'entrée de ces opérations est null.

Remarque : Ces objets ne peuvent pas être transmis dans des mappes d'objet métier car ces dernières ne peuvent pas mapper des zones depuis un objet null.

Utilisation de l'objet de séquence pour définir l'ordre des données

Certains XSD sont définis de telle sorte que l'ordre des données dans le XML a une importance significative.

Par exemple, l'ordre est important dans les XSD si le contenu est mixte. Si les données de texte apparaissent avant ou après un élément, la signification peut être différente que si elles apparaissent dans un autre emplacement. Pour ces situations, le SDO génère un objet connu sous le nom de Séquence, qui est utilisé pour définir les données d'une manière ordonnée.

Les séquences SDO ne doivent pas être confondues avec les séquences XSD. Les séquences XSD ne sont que des groupes de modèles mis à plat avant la génération du modèle SDO. La présence d'une séquence XSD n'a pas de rapport avec la présence d'une séquence SDO.

Avec les conditions suivantes, un XSD dans une séquence SDO est généré :

Un type complexe avec du contenu mixte :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://MixedContent"
 targetNamespace="http://MixedContent">
  <xsd:complexType name="MixedContent" mixed="true">
    <xsd:sequence>
      <xsd:element name="element1" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element2" type="xsd:string" minOccurs="0"/>
      <xsd:element name="element3" type="xsd:string" minOccurs="0"/>
```

```
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="MixedContent" type="tns:MixedContent"/>
</xsd:schema>
```

Un schéma comportant 1 ou plusieurs balises <any/> :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyElemAny"
 targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <xsd:any/>
      <xsd:element name="marker1" type="xsd:string"/>
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Un tableau de groupes de modèles (tous, choix, séquence ou référence de groupe avec maxOccurs > 1):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence maxOccurs="3">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Une balise <all/> de maxOccurs <= 1 contenant plusieurs éléments :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://All">
  <xsd:complexType name="All">
    <xsd:all>
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:schema>
```

Vous trouverez des informations spécifiques sur l'utilisation de <any/> avec une séquence dans la rubrique référencée en bas de cette page. Les informations d'ordre général qui suivent dans le reste de cette section expliquent comment travailler avec les autres conditions de séquence, mais s'appliquent également à <any/>.

Concepts associés

«Comment savoir si mon objet de données possède une séquence ?»
Deux API simples permettent de déterminer si un objet de données est mis en séquence : DataObject noSequence et DataObject withSequence.

«Pourquoi dois-je savoir si un objet de données possède une séquence ?»
Si vous travailler sur un objet de données comportant une séquence, il est important de connaître l'ordre dans lequel les données sont définies. Vous devez donc faire attention à l'ordre dans lequel les données sont définies.

«Comment utiliser des contenus mixtes ?», à la page 221
Pour les contenus mixtes, la séquence comporte une API spécifique pour ajouter du texte : addText(...).

«Comment utiliser un tableau de groupes de modèles ?», à la page 221
Un tableau de groupes de modèles est créé lorsque la valeur maxOccurs d'un groupe de modèles est > 1.

Comment savoir si mon objet de données possède une séquence ? :

Deux API simples permettent de déterminer si un objet de données est mis en séquence : DataObject noSequence et DataObject withSequence.

Vous pouvez utiliser DataObject noSequence et DataObject withSequence de la manière indiquée dans l'exemple suivant :

```
DataObject noSequence = ...
DataObject withSequence = ...

// Affiche la valeur faux
System.out.println(noSequence.getType().isSequenced());

// Affiche la valeur vrai
System.out.println(withSequence.getType().isSequenced());

// Affiche la valeur vrai
System.out.println(noSequence.getSequence() == null);

// Affiche la valeur faux
System.out.println(withSequence.getSequence() == null);
```

Pourquoi dois-je savoir si un objet de données possède une séquence ? :

Si vous travailler sur un objet de données comportant une séquence, il est important de connaître l'ordre dans lequel les données sont définies. Vous devez donc faire attention à l'ordre dans lequel les données sont définies.

Un objet de données qui n'est pas mis en séquence permet l'accès à un ensemble dans un ordre aléatoire. Le fonctionnement est identique à un mappage dans lequel toutes les clés sont définies sur les mêmes valeurs. L'ordre dans lequel les clés sont définies n'a pas d'importance, les données au sein du mappage étant identiques et étant sérialisées en XML d'une manière identique.

Lorsqu'un objet de données est mis en séquence, l'ordre dans lequel les données ont été définies est enregistré dans la séquence, comme s'il s'agissait d'ajouter des données à une liste. Ainsi, deux manières d'accéder aux données sont possibles : par paires nom/valeur (les API d'objet de données) et selon l'ordre dans lequel elles ont été définies (les API de séquence). Vous pouvez utiliser les API d'objet de données set(...) ou de séquence add(...) pour conserver la structure. Cet ordre a un incidence sur la manière dont le XML est sérialisé.

Prenons par exemple le XSD de la balise <all/> ci-dessous. Lorsque les méthodes set sont appelées dans l'ordre suivant, le XML suivant est produit lorsqu'il est sérialisé :

```
DataObject all = ...
all.set("element1", "foo");
all.set("element2", "bar");

<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:p="http://All">
  <element1>foo</element1>
  <element2>bar</element2>
</p:All>
```

Si, à la place, les méthodes set sont appelées dans l'ordre inverse, le XML suivant est produit lorsque l'objet métier est sérialisé :

```
DataObject all = ...
all.set("element2", "bar");
all.set("element1", "foo");

<?xml version="1.0" encoding="UTF-8"?>
<p:All xsi:type="p:All"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:p="http://All">
  <element2>bar</element2>
  <element1>foo</element1>
</p:All>
```

Si l'ordre de la séquence doit être modifié, la classe de la séquence a les méthodes add, remove et move pour permettre à l'utilisateur de modifier l'ordre de la séquence.

Comment utiliser des contenus mixtes ? :

Pour les contenus mixtes, la séquence comporte une API spécifique pour ajouter du texte : addText(...).

Toutes les autres API fonctionnent de la même manière avec du texte comme avec les propriétés. L'API getProperty(int) renverra la valeur null pour les données de texte avec des contenus mixtes. L'exemple suivant de code de contenu mixte peut être utilisé pour imprimer tout le texte avec des contenus mixtes depuis un objet de données :

```
DataObject mixedContent = ...
Sequence seq = mixedContent.getSequence();

for (int i=0; i < seq.size(); i++)
{
    Property prop = seq.getProperty(i);
    Object value = seq.getValue(i);

    si (prop == null)
    {
        System.out.println("Found mixed content text: "+value);
    }
    else
    {
        System.out.println("Found Property "+prop.getName()+": "+value);
    }
}
```

Comment utiliser un tableau de groupes de modèles ? :

Un tableau de groupes de modèles est créé lorsque la valeur maxOccurs d'un groupe de modèles est > 1.

Les groupes de modèles étant mis à plat et n'étant pas exprimés dans un objet de données, les propriétés au sein du groupe de modèles deviennent des propriétés à cardinalité multiple et leurs méthodes isMany() renvoient la valeur vrai si elles ne l'ont pas déjà. Leurs facettes minOccurs et maxOccurs sont alors multipliées par celles du groupe de modèles qui les contient. Le choix multipliera la facette maxOccurs de la même manière que les autres groupes de modèles, mais utilisera toujours 0 comme valeur de multiplication pour minOccurs, car toutes les données dans le choix peuvent ne pas être sélectionnées.

Par exemple, le XSD suivant comporte un tableau de groupes de modèles :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:sequence minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
   minOccurs="0" maxOccurs="3"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Comme indiqué, **element1** et **element2** seront maintenant une cardinalité multiple et un mécanisme d'accès get(...) renverra donc une liste. **Element1** a par défaut la valeur minOccurs 1 et la valeur maxOccurs 1. **Element2** a la valeur minOccurs 0 et la valeur maxOccurs 3. Dans l'exemple suivant, leurs nouvelles valeurs minOccurs et maxOccurs seront les suivantes :

```
Data Object - ModelGroupArray
 Property[0] - element1 - minOccurs=(2*1)=2 - maxOccurs=(5*1)=5
 Property[1] - element2 - minOccurs=(2*0)=0 - maxOccurs=(5*3)=15
```

Si le type était Choix :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ModelGroupArray">
  <xsd:complexType name="ModelGroupArray">
    <xsd:choice minOccurs="2" maxOccurs="5">
      <xsd:element name="element1" type="xsd:string"/>
      <xsd:element name="element2" type="xsd:string"
   minOccurs="0" maxOccurs="3"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

La valeur minOccurs suivante serait générée, en raison de l'exclusion du choix indiquant que seul **element1** puisse être extrait à chaque fois ou que seul **element2** puisse être extrait à chaque fois, et les deux doivent pouvoir avoir 0 occurrence pour réussir la validation :

```
DataObject - ModelGroupArray
 Property[0] - element1 - minOccurs=(0*1)=0 - maxOccurs=(5*1)=5
 Property[1] - element2 - minOccurs=(0*0)=0 - maxOccurs=(5*3)=15
```

Utilisation de tout type de données

Cette section fournit des techniques de programmation permettant d'utiliser tout type de données.

Concepts associés

«Utilisation de AnySimpleType pour les types simples»
AnySimpleType est traité de la même manière que les autres types simples (chaîne, int, booléen, etc.) par les API SDO.

«Utilisation de AnyType pour les types complexes», à la page 225
La balise anyType est traitée de la même manière que les autres types complexes par les API SDO.

«Utilisation de la balise Any pour définir des éléments globaux de types complexes», à la page 227
Vous pouvez utiliser la balise <any/> pour définir des éléments globaux sur un type complexe.

«Utilisation de AnyAttribute pour définir les attributs globaux de types complexes» , à la page 230
La balise <anyAttribute/> permet de définir n'importe quel ensemble d'attributs globaux sur un type complexe.

Utilisation de AnySimpleType pour les types simples :

AnySimpleType est traité de la même manière que les autres types simples (chaîne, int, booléen, etc.) par les API SDO.

Les seules différences entre anySimpleType et les autres types simples sont dans ses données d'instance et la sérialisation/désérialisation. Elles doivent être des concepts internes pour les objets métier uniquement, et elles sont utilisées pour déterminer si les données mappées vers ou depuis le champ sont valides. Si une méthode set(...) devait être appelée sur un type de chaîne, les données seraient d'abord converties en une chaîne, et les données d'origine seraient perdues :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://StringType">
  <xsd:complexType name="StringType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject stringType = ...

 // Définir les données sur une chaîne
 stringType.set("foo", "bar");

 // Les données d'instance seront toujours du type chaîne, quelles que soient
les données définies
 // Affiche "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());

 // Définir les données sur un nombre entier
 stringType.set("foo", new Integer(42));

 // Les données d'instance seront toujours du type chaîne, quelles que soient
les données définies
 // Affiche "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());
```

Un élément anySimpleType à la place ne perd pas le type de données d'origine de ce qui est défini :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://AnySimpleType">
  <xsd:complexType name="AnySimpleType">
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:anySimpleType"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject anySimpleType = ...

 // Définir les données sur une chaîne
 stringType.set("foo", "bar");

 // Les données d'instance seront toujours du type date utilisé dans l'ensemble
 // Affiche "java.lang.String"
 System.out.println(stringType.get("foo").getClass().getName());

 // Définir les données sur un nombre entier
 stringType.set("foo", new Integer(42));

 // Les données d'instance seront toujours du type date utilisé dans l'ensemble
 // Affiche "java.lang.Integer"
 System.out.println(stringType.get("foo").getClass().getName());
```

Ce type de données est également préservé lors de la sérialisation et désérialisation par xsi:type. En conséquence, à chaque fois que vous sérialisez un élément anySimpleType, il aura un xsi:type qui correspond à celui défini dans la spécification SDO en fonction de son type Java :

Dans l'exemple suivant, vous sérialisez l'objet métier ci-dessus de telle sorte que les données ressembleront à :

```
<?xml version="1.0" encoding="UTF-8"?>
<p:StringType xsi:type="p:StringType"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:p="http://StringType">
  <foo xsi:type="xsd:int">42</foo>
<p:StringType></p:StringType>
```

Le xsi:type sera utilisé lors de la désérialisation pour charger les données comme classe d'instance Java appropriée. Si aucun xsi:type n'est spécifié, le type de désérialisation par défaut sera une chaîne.

Pour les autres types simples, déterminer la mappabilité est une constante. Par exemple, un élément booléen peut toujours mapper une chaîne. AnySimpleType peut contenir n'importe quel type simple, mais un mappage peut être possible ou non, en fonction des données d'instance dans le champ.

Utilisez le type de propriété URI et Nom pour déterminer si une propriété est du type anySimpleType. Il s'agira de "commonj.sdo" et "Object". Pour déterminer si des données sont valides pour être insérées dans anySimpleType, vérifiez s'il ne s'agit pas d'une instance d'un objet de données. Toutes les données pouvant être représentées sous la forme d'une chaîne et n'étant pas un objet de données peuvent être définies dans un champ anySimpleType.

Les règles de mappage sont donc les suivantes :

- anySimpleType peut toujours être mappé sur anySimpleType.
- n'importe quel autre type simple peut toujours être mappé sur anySimpleType.

- anySimpleType peut toujours être mappé sur une chaîne car tous les types simples doivent pouvoir être convertis en une chaîne.
- anySimpleType peut ou ne peut pas être mappé sur un des autres types simples, en fonction de sa valeur dans l'objet métier. Cela signifie que ce mappage ne peut pas être déterminé au moment de la conception, mais uniquement lors de l'exécution.

Information associée

Affectation depuis et vers xs:any

Utilisation de AnyType pour les types complexes :

La balise anyType est traitée de la même manière que les autres types complexes par les API SDO.

Les seules différences entre anyType et les autres types complexes sont dans leurs données d'instance et la sérialisation/désérialisation, qui doivent être des concepts internes pour l'objet métier uniquement, et déterminant si les données mappées vers ou depuis le champ sont valides. Les types complexes sont limités à un type unique : Client, Adresse, etc. La balise anyType, cependant, permet n'importe quel objet de données quel que soit le type. Si maxOccurs > 1, chaque objet de données de la liste peut être d'un type différent.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://AnyType">
  <xsd:complexType name="AnyType">
    <xsd:sequence>
      <xsd:element name="person" type="xsd:anyType"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://Customer">
  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://Employee" targetNamespace="http://Employee">
  <xsd:complexType name="Employee">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"/>
 </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

 DataObject anyType = ...
 DataObject customer = ...
 DataObject employee = ...

 // Définir la personne sur Customer
 anyType.set("person", customer);

 // Les données d'instance seront un client
 // Affiche "Customer"
 System.out.println(anyType.getDataObject("person").getName());
```

```
// Définir la personne sur Employee
anyType.set("person", employee);

// Les données d'instance seront un employé
// Affiche "Employee"
System.out.println(anyType.getDataObject("person").getName());
```

Comme anySimpleType, anyType utilise l'élément xsi:type lors de la sérialisation afin d'assurer que le type d'objet de données voulu est conservé lorsqu'il est désérialisé. Ainsi, si vous le définissez sur "Customer", le XML se présente comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:customer="http://Customer"
    xmlns:p="http://AnyType">
  <person xsi:type="customer:Customer">
    <name>foo</name>
  </person>
</p:AnyType>
```

Et, si vous le définissez sur "Employee" :

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:employee="http://Employee"
    xmlns:p="http://AnyType">
  <person xsi:type="employee:Employee">
    <id>foo</id>
  </person>
</p:AnyType>
```

La balise AnyType permet également de définir des valeurs de type simple par le biais d'objets de données encapsuleurs. Ces objets de données encapsuleurs possèdent une propriété unique appelée "value" (élément) qui contient la valeur de type simple. Les API SDO ont été écrasées pour encapsuler et désencapsuler automatiquement ces objets de données de types simples et encapsuleurs lorsque les API de <Type>get/set<> sont utilisées. Les API get/set de transtypage non-type n'effectueront pas cet encapsulage.

```
DataObject anyType = ...

// Appeler une API de <Type> set sur une propriété anyType entraîne la création
// automatique d'un objet de données encapsuleur
anyType.setString("person", "foo");

// Les API get/set classiques ne sont pas écrasées, et renverront donc
// l'objet de données encapsuleur
DataObject wrapped = anyType.get("person");

// L'objet de données encapsulé aura la propriété "value"
// Affiche "foo"
System.out.println(wrapped.getString("value"));

// L'API de <Type> get désencapsulera automatiquement l'objet de données
// Affiche "foo"
System.out.println(anyType.getString("person"));
```

Lorsque l'objet de données encapsuleur est sérialisé, il est sérialisé de la même manière qu'un mappage anySimpleType de classes d'instance Java en types XSD dans le champ xsi:type. Ce paramètre doit donc être sérialisé de la manière suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<p:AnyType xsi:type="p:AnyType"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:p="http://AnyType">
  <person xsi:type="xsd:string">foo</person>
</p:AnyType>
```

Si aucun élément xsi:type n'est donné ou si un élément xsi:type incorrect est donné, une exception est émise. En plus de l'encapsulage automatique, l'encapsuleur peut être créé manuellement pour être utilisé avec l'API set() via BOFactory createDataTypeWrapper(Type, Object), où Type est le type simple de SDO des données à encapsuler et Object représente les données à encapsuler.

```
Type stringType = boType.getType("http://www.w3.org/2001/XMLSchema", "string");
DataObject stringType = boFactory.createByMessage(stringType, "foo");
```

Pour déterminer si un objet de données est du type encapsuleur, l'élément BOType isDataTypeWrapper(Type) peut être appelé.

```
DataObject stringType = ...
boolean isWrapper = boType.isDataTypeWrapper(stringType.getType());
```

Pour les autres types complexes, pour pouvoir déplacer les données d'un champ à l'autre, les données doivent être du même type. La balise AnyType peut contenir n'importe quel type complexe, mais un déplacement direct sans mappage peut être basé sur les données d'instance dans le champ ou non.

Vous pouvez utiliser l'URI et le Nom du type de propriété pour déterminer si une propriété est du type anyType. Il s'agira de "commonj.sdo" et "DataObject". Toutes les données sont valides pour être insérées dans une balise anyType. Les règles de mappage sont donc les suivantes :

- anyType peut toujours être mappé sur anyType.
- n'importe quel type complexe peut toujours être mappé sur anyType.
- n'importe quel type simple peut toujours être mappé sur anyType.
- anyType peut ou ne peut pas être mappé sur un des autres types simples ou complexes, en fonction de sa valeur dans l'instance d'objet métier. Cela signifie que ce mappage ne peut pas être déterminé au moment de la conception, mais uniquement au moment de l'exécution.

Utilisation de la balise Any pour définir des éléments globaux de types complexes :

Vous pouvez utiliser la balise <any/> pour définir des éléments globaux sur un type complexe.

Avec une occurrence de la balise any, les méthodes DataObject Type isOpen() et isSequenced() renvoient la valeur vrai. Si la valeur de maxOccurs est > 1 sur une balise any, cela n'a aucune incidence sur la structure de l'objet de données ; elle est utilisée uniqement comme information lors de la validation. De la même manière, l'occurrence de balises any multiples dans un type ne modifie pas la structure de l'objet de données ; elles sont utilisées uniquement pour valider l'emplacement des données ouvertes qui ont été définies.

Concepts associés

«Comment savoir si mon objet de données possède une balise ?»
Vous pouvez déterminer facilement si des valeurs sont définies au sein d'un objet de données en vérifiant leurs propriétés pour voir si des propriétés ouvertes sont des attributs.

«Comment obtenir/définir des valeurs ?»
Vous pouvez exécuter une instruction get sur des données qui ont été définies dans un champ de la même manière que pour une autre valeur d'élément si le nom est connu.

«Quels sont les mappages de données valides pour la valeur Any ?», à la page 230
Une balise <any/> est un ensemble de paires nom/valeur. Le seul mappage valide pouvant être déterminé au moment de la conception pour <any/> est une autre balise <any/> ou anyType ayant la même valeur maxOccurs.

Comment savoir si mon objet de données possède une balise ? :

Vous pouvez déterminer facilement si des valeurs sont définies au sein d'un objet de données en vérifiant leurs propriétés pour voir si des propriétés ouvertes sont des attributs.

L'objet de données ne possède pas de mécanisme permettant de déterminer si un type d'objet de données comporte des balises. Les objets de données possèdent uniquement le concept "ouvert" qui s'applique à la balise any et à la balise anyAttribute, et qui permet d'ajouter librement des propriétés. Alors que la présence d'une balise implique qu'un objet de données a la valeur isOpen() = vrai et isSequenced() = vrai, il peut comporter uniquement une balise anyAttribute et une des raisons pour lesquelles il est mis en séquence décrite dans la rubrique Séquences. L'exemple suivant explique ces concepts :

```
DataObject dobj = ...

// Vérifiez si le type est "ouvert" ; dans le cas contraire, aucune valeur
// ne peut être définie pour cet objet de données.
boolean isOpen = dobj.getType().isOpen();

si (!isOpen) renvoie la valeur faux ;  // Aucune valeur n'est définie pour cet objet de données

// Les propriétés ouvertes sont ajoutées à la liste des propriétés de l'instance, mais pas
// la liste des propriétés. Ainsi, comparer leurs tailles permet de déterminer facilement
// si des données ouvertes sont définies
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// Si elles sont égales, aucun contenu ouvert n'est défini
si (instancePropertyCount == definedPropertyCount) renvoie la valeur faux ;

// Vérifiez les propriétés du contenu ouvertes pour déterminer si certaines d'entre
elles sont des éléments
for (int i=definedPropertyCount; i < instancePropertyCount; i++)
{
    Property prop = (Property)dobj.getInstanceProperties().get(i);
    si (boXsdHelper.isElement(prop))
    {
        renvoie la valeur vrai ;  // Une valeur any a été trouvée
    }
}

renvoie la valeur faux ;  // Aucune valeur n'est définie
```

Comment obtenir/définir des valeurs ? :

Vous pouvez exécuter une instruction get sur des données qui ont été définies dans un champ de la même manière que pour une autre valeur d'élément si le nom est connu.

Vous pouvez envoyer une instruction get avec la balise XPath "<name>" pour la résoudre. Si le nom est inconnu, il est possible de trouver la valeur en vérifiant les propriétés de l'instance comme ci-dessus. S'il y a plusieurs balises any, ou une balise any avec maxOccurs > 1, la séquence de l'objet de données devra être utilisée à la place s'il est important de déterminer quelle balise any est à l'origine des données.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyElemAny"
 targetNamespace="http://AnyElemAny">
  <xsd:complexType name="AnyElemAny">
    <xsd:sequence>
      <!-- Handle all these any one way -->
      <xsd:any maxOccurs="3"/>
      <xsd:element name="marker1" type="xsd:string"/>
      <!-- Handle this any in another -->
      <xsd:any/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

La balise <any/> entraînant la mise en séquence de l'objet de données, il est possible de déterminer quelle valeur any a été définie en vérifiant dans la séquence la position des propriétés any.

Vous pouvez déterminer à quelle balise any les données d'instance appartiennent pour le XSD suivant en utilisant le code suivant :

```
DataObject anyElemAny = ...
Seqeuence seq = anyElemAny.getSequence();

// Jusqu'à ce que l'élément marker1 ait été trouvé, toutes les données ouvertes
// trouvées appartiennent à la première balise any
boolean foundMarker1 = false;

for (int i=0; i<seq.size(); i++)
{
    Property prop = seq.getProperty(i);

    // Vérifiez si la propriété est une propriété ouverte
    si (prop.isOpenContent())
    {
        si (!foundMarker1)
        {
            // Doit être la première balise any car elle survient
            // avant l'élément marker1
            System.out.println("Found first any data: "+seq.getValue(i));
        }
        else
        {
            // Doit être la seconde balise any car elle survient
            // après l'élément marker1
            System.out.println("Found second any data: "+seq.getValue(i));
         }
    }
    else
    {
        // Doit être l'élément marker1
```

```
            System.out.println("Found marker1 data: "+seq.getValue(i));
            foundMarker1 = true;
        }
    }
```

Définir une valeur <any/> est effectué en créant une propriété d'élément global et en ajoutant cette valeur à la séquence.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://GlobalElems"
 targetNamespace="http://GlobalElems">
  <xsd:element name="globalElement1" type="xsd:string"/>
  <xsd:element name="globalElement2" type="xsd:string"/>
</xsd:schema>

 DataObject anyElemAny = ...
 Seqeuence seq = anyElemAny.getSequence();

 // Obtenir la propriété de l'élément global pour globalElement1
 Property globalProp1 = boXsdHelper.getGlobalProperty(http://GlobalElems,
 "globalElement1", true);

 // Obtenir la propriété de l'élément global pour globalElement2
 Property globalProp2 = boXsdHelper.getGlobalProperty(http://GlobalElems,
 "globalElement2", true);

 // Ajouter les données à la séquence pour la première balise any
 seq.add(globalProp1, "foo");
 seq.add(globalProp1, "bar");

 // Ajouter les données pour le marker1
 seq.add("marker1", "separator");  // ou anyElemAny.set("marker1", "separator")

 // Ajouter les données à la séquence pour la seconde balise any
 seq.add(globalProp2, "baz");

 // Il est maintenant possible d'accéder aux données avec une instruction get
 System.out.println(dobj.get("globalElement1"); // Affiche "[foo, bar]"
 System.out.println(dobj.get("marker1");        // Affiche "separator"
 System.out.println(dobj.get("globalElement2"); // Affiche "baz"
```

Quels sont les mappages de données valides pour la valeur Any ? :

Une balise <any/> est un ensemble de paires nom/valeur. Le seul mappage valide pouvant être déterminé au moment de la conception pour <any/> est une autre balise <any/> ou anyType ayant la même valeur maxOccurs.

Individuellement, les valeurs contenues dans une instance d'un objet de données pour la balise any sont des types complexes de base respectant les règles d'un mappage de type complexe. Certains de ces types complexes peuvent être des types simples encapsulés, et ils suivront les règles du mappage de type simple.

Utilisation de AnyAttribute pour définir les attributs globaux de types complexes :

La balise <anyAttribute/> permet de définir n'importe quel ensemble d'attributs globaux sur un type complexe.

Comme pour la balise <any/>, l'occurrence de la balise <anyAttribute/> entraîne le renvoi par la méthode DataObject Type isOpen() de la valeur true. Toutefois,

contrairement à la balise <any/>, <anyAttribute/> n'implique pas le séquencement de l'objet données, car les attributs de XSD ne sont pas des constructions ordonnées.

Concepts associés

«Comment savoir si mon objet de données possède une balise AnyAttribute ?»
Vous pouvez aisément déterminer si des instances d'un objet données comportent un ensemble de valeurs anyAttribute en vérifiant leurs propriétés pour savoir si les propriétés ouvertes représentent des attributs.

«Comment obtenir/définir des valeurs AnyAttribute ?», à la page 232
Définir une valeur <anyAttribute/> est effectué de la même manière que pour une balise <any/>, mais un attribut global est utilisé à la place d'un élément global.

«Quels sont les mappages de données valides pour la valeur AnyAttribute ?», à la page 232
La balise AnyAttribute est similaire à la balise any, et correspond à un ensemble de paires nom/valeur. En conséquence, le seul mappage valide pour anyAttribute est une autre balise anyAttribute.

Comment savoir si mon objet de données possède une balise AnyAttribute ? :

Vous pouvez aisément déterminer si des instances d'un objet données comportent un ensemble de valeurs anyAttribute en vérifiant leurs propriétés pour savoir si les propriétés ouvertes représentent des attributs.

L'objet données ne prévoit pas de mécanisme pour déterminer si un type d'objet données inclut une balise anyAttribute. Seuls les objets données comportent un concept d'ouverture qui s'applique aux balises any et <anyAttribute/> et qui permet l'ajout de propriétés supplémentaires. S'il est vrai que si un objet données a défini isOpen() = true et isSequenced() = false, il doit inclure une balise anyAttribute, si isOpen() = true et isSequenced() = true, le type d'objet données pouvant ou non inclure une balise anyAttribute.

L'objet données fournit des méthodes d'interrogation des métadonnées pour répondre à l'aide d'un programme à cette question et bien d'autres sur la structure XSD qui a servi à le générer. Le modèle InfoSet peut être interrogé pour déterminer si nécessaire l'existence de la balise anyAttribute. Parce que la balise anyAttribute est unique et que sa valeur peut être ou non true, les objets métier fournissent également une méthode BOXSDHelper hasAnyAttribute(Type) pour déterminer si la définition d'un attribut ouvert sur cet objet données produira un résultat valide. L'exemple de code suivant illustre ces concepts :

```
DataObject dobj = ...

// Vérifiez si le type est ouvert. S'il ne l'est pas, aucune
// valeur anyAttribute ne peut y être définie.
boolean isOpen = dobj.getType().isOpen() ;

si (!isOpen) return false ;  // Aucune valeur anyAttribute définie

// Les propriétés ouvertes sont ajoutées à la liste des propriétés de l'instance,
// mais pas la liste des propriétés.
// Par conséquent, la comparaison de leurs tailles peut permettre
de déterminer facilement
// si des données ouvertes sont définies
int instancePropertyCount = dobj.getInstanceProperties().size();
int definedPropertyCount = dobj.getType().getProperties().size();

// Si leur taille est identique, aucun contenu ouvert n'est défini
si (instancePropertyCount == definedPropertyCount) return false ;
```

```
 // Vérifiez les propriétés du contenu ouvert pour déterminer
l'une d'elles constituent des attributs
 pour (int i=definedPropertyCount; i<instancePropertyCount; i++)
 {
     Property prop = (Property)dobj.getInstanceProperties().get(i);
     si (boXsdHelper.isAttribute(prop))
     {
         return true ;  // Valeur anyAttribute trouvée
     }
 }

 return false ;  // Aucune valeur anyAttribute définie
```

Comment obtenir/définir des valeurs AnyAttribute ? :

Définir une valeur <anyAttribute/> est effectué de la même manière que pour une balise <any/>, mais un attribut global est utilisé à la place d'un élément global.

Exécuter une instruction get sur des données qui ont été définies dans un champ anyAttribute peut être effectué de la même manière que pour une autre valeur d'attribut si le nom est connu. Vous pouvez envoyer une instruction get avec la balise XPath "@<name>" pour la résoudre. Si le nom est inconnu, vous pouvez utiliser le code ci-dessus pour itérer les valeurs et y accéder une par une.
L'exemple de code suivant montre comment procéder :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="http://AnyAttrOnlyMixed"
 targetNamespace="http://AnyAttrOnly">
  <xsd:complexType name="AnyAttrOnly">
    <xsd:sequence>
      <xsd:element name="element" type="xsd:string"/>
    </xsd:sequence>
    <xsd:anyAttribute/>
  </xsd:complexType>
</xsd:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://GlobalAttrs">
  <xsd:attribute name="globalAttribute" type="xsd:string"/>
</xsd:schema>

 Data Object dobj = ...

 // Obtenir la propriété de l'attribut global qui va être défini
 Property globalProp = boXsdHelper.getGlobalProperty(http://GlobalAttrs,
 "globalAttribute", false);

 // Définir la valeur sur l'objet de données, comme n'importe quelle autre donnée
 dobj.set(globalProp, "foo");

 // Il est maintenant possible d'accéder aux données avec une instruction get
 System.out.println(dobj.get("@globalAttribute")); // Affiche "foo"
```

Quels sont les mappages de données valides pour la valeur AnyAttribute ? :

La balise AnyAttribute est similaire à la balise any, et correspond à un ensemble de paires nom/valeur. En conséquence, le seul mappage valide pour anyAttribute est une autre balise anyAttribute.

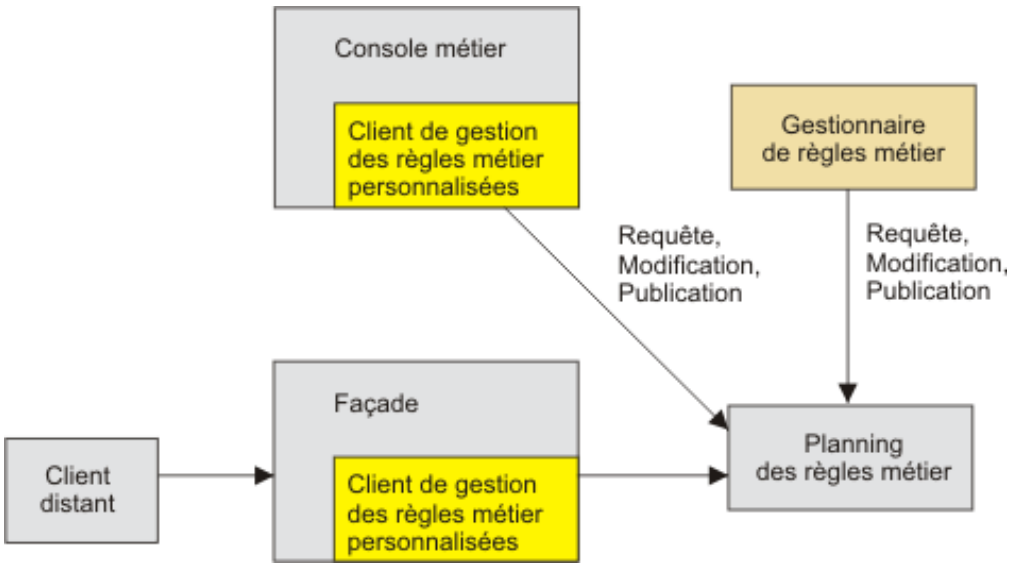
Individuellement, les valeurs contenues dans les données anyAttribute sont des types simples de base respectant les règles du mappage de type simple

Programmation de la gestion des règles métier

Des classes de gestion des règles métier sont fournies pour permettre de créer des clients de gestion personnalisés ou d'automatiser les changements apportés aux règles métier.

Les classes de gestion des règles métier peuvent être utilisées dans une application Web, où elles sont combinées à d'autres capacités de gestion pour des processus métier ou des tâches manuelles, afin de gérer tous les composants d'un même client. Vous pouvez utiliser tout client de gestion personnalisé avec l'application Web Business Rules Manager contenue dans WebSphere Process Server. Les classes peuvent également être utilisées pour l'automatisation des modifications apportées aux règles métier au sein d'une application. Par exemple, certaines règles métier peuvent être modifiées si les résultats d'un processus métier utilisant ces règles dépassent un seuil ou une limite spécifique.

Les classes de gestion des règles métier doivent être utilisées dans une application installée surWebSphere Process Server. Les classes n'incluent pas d'interface distante, mais elles peuvent être encapsulées dans une façade, qui est ensuite exposée via un protocole spécifique, à des fins d'exécution à distance.



Ce guide de programmation se compose de deux sections principales et d'une annexe. La première section explique le modèle de programmation, et indique comment utiliser les différentes classes. Des diagrammes de classes sont fournis pour illustrer les relations existant entre les classes. La deuxième section contient des exemples d'utilisation des classes pour l'exécution d'opérations telles que la recherche de groupes de règles métier, la planification de la destination d'une nouvelle règle, ou encore la modification d'un ensemble de règles ou d'une table de décision. L'annexe contient des classes supplémentaires, qui ont été utilisées dans les exemples pour simplifier des opérations courantes, et d'autres exemples de création de requêtes complexes servant à rechercher des groupes de règles métier en utilisant des caractères génériques.

Ce guide de programmation consacré aux classes est également disponible au format HTML Javadoc inclus dans WebSphere Process Server v6.1 et dans l'environnement de test de WebSphere Integration Developer v6.1. Cette documentation Javadoc est figure dans le répertoire ${Répertoire d'installation de WebSphere Process Server}\web\apidocs ou dans ${Répertoire d'installation de

WebSphere Integration Developer}\runtimes\bi_v61\web\apidocs. Les packages com.ibm.wbiserver.brules.mgmt.* contiennent toutes les informations.

Concepts associés

«Modèle de programmation»
Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Exemples
Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Référence associée

Classes d'opérations communes
Cette section contient des classes supplémentaires, qui ont été utilisées dans les exemples pour simplifier des opérations communes.

Modèle de programmation

Les règles métier de WebSphere Business Integration sont créées à l'aide de deux outils de création différents, et sont émises par le dispositif d'exécution de règles. Tous trois partagent le même modèle d'artefacts de règles métier.

Le partage d'un même modèle a été jugé essentiel pour des raisons de maintenance future, et également dans le but d'offrir à l'utilisateur final un modèle de programmation cohérent. Le partage de ce modèle a nécessité des compromis entre les besoins d'outils, l'exécution et la création : en effet, tous ces aspects possèdent leurs propres exigences en fonction de leur environnement respectif ; or, ces exigences entraient parfois en conflit. Les artefacts décrits ci-dessous en tant que partie intégrante du modèle de programmation global représentent un équilibre entre toutes les exigences de ces environnements différents.

La modification des règles métier est limitée aux seuls éléments définis à l'aide de modèles dans les ensembles de règles, dans les tables de décision et dans la table de sélection des opérations (dates d'entrée en vigueur et cibles). La création de nouveaux ensembles de règles et de nouvelles tables de décisions n'est prise en charge que via la copie d'un ensemble de règles ou d'une table de décision existant(e). Le composant de groupe de règle métier lui-même ne peut pas être créé dynamiquement lors de l'exécution, à l'exception des propriétés définies par l'utilisateur et des valeurs de description. Pour apporter les modifications requises au composant (ajout d'une nouvelle opération, par exemple), vous devez utiliser WebSphere Integration Developer, puis redéployer ces modifications ou les réinstaller sur le serveur.

Concepts associés

«Groupe de règles métier», à la page 236
La classe BusinessRuleGroup représente le composant de groupe de règles métier. Cette classe peut être considérée comme l'objet racine contenant les ensembles de règles et les tables de décision.

«Propriétés de groupes de règles métier», à la page 238
Les propriétés des groupes de règles métier servent à gérer ces groupes. Les propriétés définies dans les groupes de règles métier peuvent être utilisées dans les requêtes, pour renvoyer uniquement un sous-ensemble de groupes de règles métier à afficher puis à modifier.

«Opération», à la page 239
Les opérations représentent le point de départ d'accès aux ensembles de règles et aux tables de décisions à modifier. Les opérations d'un groupe de règles métier correspondent aux opérations répertoriées dans le langage WSDL associé au composant de groupes de règles métier.

«Règle métier», à la page 242
Les classes RuleSet et DecisionTable sont basées sur une classe générique BusinessRule et contiennent des méthodes fournissant les informations disponibles dans les ensembles de règles et les tables de décision.

«Ensemble de règles», à la page 244
Un ensemble de règles constitue un type de règle métier. Les ensembles de règles sont généralement utilisés lorsque plusieurs règles doivent être exécutées sur la base de différentes valeurs conditionnelles. Les ensembles de règles se composent d'un bloc de règles et de modèles de règles. Le bloc de règles (`RuleBlock`) contient les différentes règles if-then et action qui composent la logique de l'ensemble de règles.

table de décision
Les tables de décision représentent un autre type de règle métier que vous pouvez gérer et modifier. Elles sont généralement utilisées lorsque de nombreuses conditions doivent être évaluées et qu'un ensemble spécifique d'actions doivent être émises une fois les conditions remplies.

Modèles et paramètres
Les modèles inclus dans les ensembles de règles et dans les tables de décision prennent comme base une définition commune. Les modèles possèdent des paramètres et une présentation de l'utilisateur. Les valeurs de paramètres inclus dans les modèles sont définis pour permettre d'apporter des modifications à la règle une fois que celle-ci a été déployée.

Validation
Parmi les objets principaux, nombreux sont ceux qui possèdent une méthode de validation ; elle permet de vérifier si les artefacts sont corrects et complets avant leur publication.

Suivi des modifications
Pour tous les objets, vous pouvez utiliser une méthode `hasChanges` afin de vérifier si des modifications ont été apportées à l'objet et aux objets qu'il contient.

BusinessRuleManager
La classe `BusinessRuleManager` est la principale classe d'utilisation des groupes de règles, des ensembles de règles et des tables de décision.

Traitement des exceptions
Des exceptions peuvent être générées lors d'un appel de validation pour un artefact ou lors de sa publication. En cas d'erreur de validation, l'exception `ValidationException` est générée ; elle s'accompagne de la liste des problèmes rencontrés. Si un problème survient au cours de la publication car une autre transaction publie les mêmes artefacts, l'exception `ChangeConflictException` est

générée. A chaque détection de la modification d'un artefact par une autre transaction, l'exception `ChangeConflictException` est générée.

Autorisation
Les classes ne prennent en charge aucun niveau d'autorisation. L'application client utilisant les classes doit ajouter sa propre méthode d'autorisation.

Groupe de règles métier

La classe BusinessRuleGroup représente le composant de groupe de règles métier. Cette classe peut être considérée comme l'objet racine contenant les ensembles de règles et les tables de décision.

Les ensembles de règles et les tables de décision sont accessibles uniquement par le groupe de règles métier auquel elles sont associées. La classe contient des méthodes permettant d'extraire les informations liées au groupe de règles métier et d'accéder aux ensembles de règles et aux tables de décision. Les méthodes permettent d'extraire les informations suivantes :

- Espace de nom cible
- Nom de groupe de règles métier
- Nom affiché
- Synchronisation nom/nom affiché
- Description
- Fuseau horaire de présentation indiquant si les dates doivent être affichées au format UTC (temps universel coordonné) ou en local sur le système
- Opérations définies dans l'interface associée au groupe de règles métier
- Propriétés personnalisées définies dans le groupe de règles métier

Les différents ensembles de règles et tables de décision associés au groupe de règles métier sont accessibles par l'opération du groupe de règles métier.

Des méthodes permettent également de mettre à jour les informations dans le groupe de règles métier. Les informations suivantes peuvent être mises à jour via les méthodes :

- Description
- Nom affiché
- Synchronisation nom/nom affiché
- Propriétés personnalisées définies dans le groupe de règles métier

Le nom affiché du groupe de règles métier peut être défini de manière explicite ou sur la valeur du nom à l'aide de la méthode setDisplayNameIsSynchronizedToName.

Les autres valeurs ne peuvent pas être modifiées puisqu'elles font partie de la définition du composant de groupe de règles métier. Leur modification nécessiterait un redéploiement ainsi qu'une réinstallation.

La classe du groupe de règles métier offre également une méthode d'actualisation. Cette méthode effectue un appel vers la mémoire persistante ou le référentiel dans lesquels les règles métier sont stockées et renvoie le groupe de règles métier ainsi que tous les ensembles de règles et les tables de décision avec les informations conservées. Le groupe de règles métier renvoyé représente la dernière copie et l'objet précédent devient obsolète.

La méthode isShell permet de dire si la version d'une instance de groupe de règles métier est prise en charge par l'exécution en cours. Par exemple, si un client Web a été créé avec les classes de gestion de règles métier en cours, et que de nouvelles fonctions ajoutées ultérieurement au groupe de règles métier ne sont pas prises en charge par les classes, un groupe de règles métier interpréteur de commandes est créé une fois le groupe de règles métier récupéré. Cela permet au client Web de continuer à utiliser les règles métier prises en charge et à récupérer les groupes de règles métier avec des fonctions et des attributs limités. Lorsque la méthode isShell est vraie, seules les méthodes getName, getTargetNameSpace, getProperties, getPropertyValue et getProperty renvoient des valeurs. Toutes les autres méthodes conduisent à l'exception UnsupportedOperationException. Outre l'utilisation de la méthode isShell, le type de BusinessRuleGroup peut également être vérifié s'il s'agit d'une instance de BusinessRuleGroupShell, afin de déterminer si la version est prise en charge.

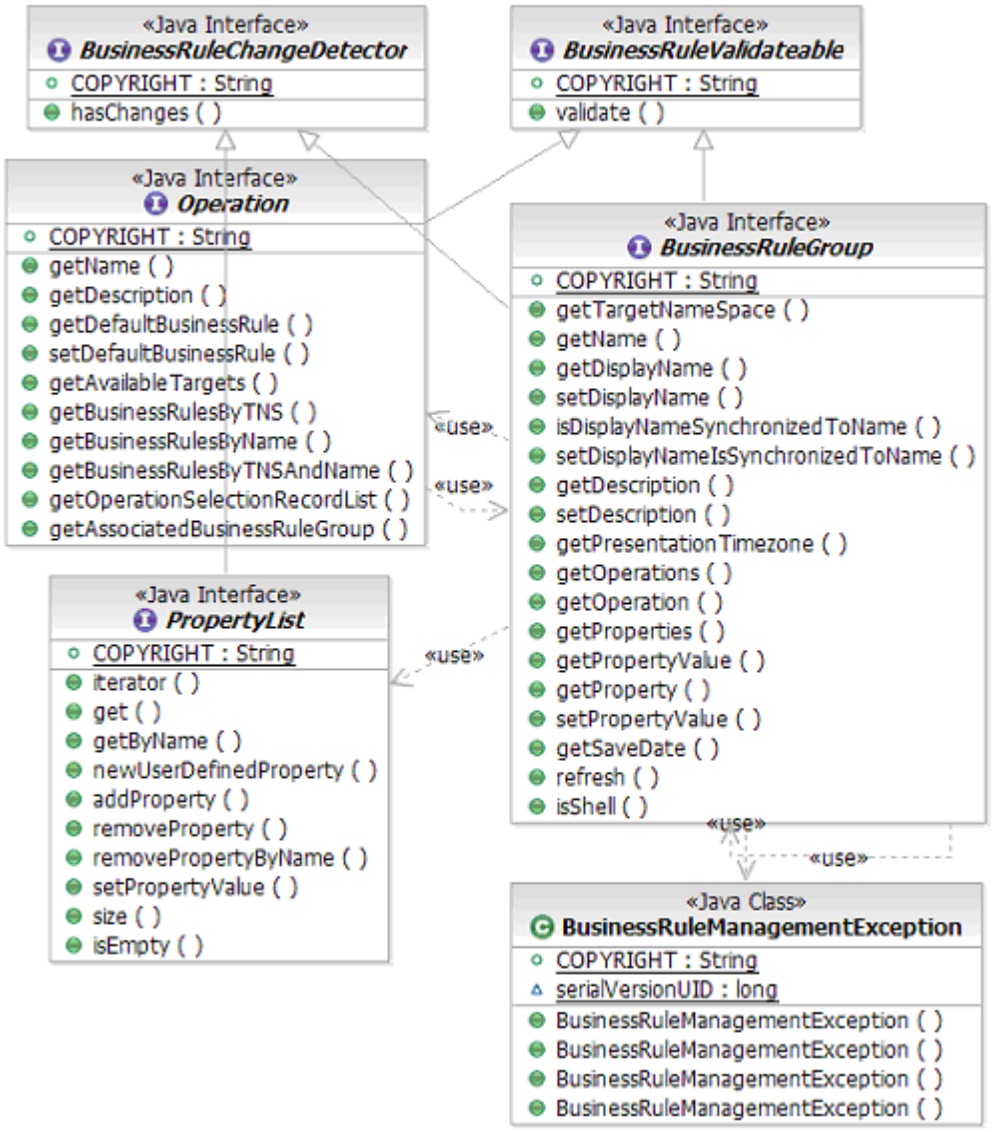


Figure 59. Diagramme de classes de BusinessRuleGroup et classes associées

Propriétés de groupes de règles métier

Les propriétés des groupes de règles métier servent à gérer ces groupes. Les propriétés définies dans les groupes de règles métier peuvent être utilisées dans les requêtes, pour renvoyer uniquement un sous-ensemble de groupes de règles métier à afficher puis à modifier.

Toutes les propriétés sont du type chaîne et sont définies en tant que paires valeur-nom. Chaque propriété ne peut être définie qu'une seule fois dans un groupe de règles métier. Pour chaque propriété définie, une valeur doit également lui être définie. La valeur de propriété peut être une chaîne vide ou de longueur zéro, mais pas NULL. Définir une propriété sur NULL revient à la supprimer.

Les propriétés d'un groupe de règles métier sont également accessibles dans un ensemble de règles ou une table de décision au moment de l'exécution. Cela permet à une valeur unique, à définir dans le groupe de règles métier, d'être utilisée au sein de plusieurs ensembles de règles ou de tables de décision du groupe de règles métier. Seules les propriétés définies dans le groupe de règles métier sont disponibles pour les ensembles de règles et les tables de décision joints.

Il existe deux types de propriétés, système et définies par l'utilisateur. Le nombre de propriétés système ou de propriétés définies par l'utilisateur n'est pas limité dans un groupe de règles métier. Les propriétés système permettent de détenir des informations spécifiques liées à un composant telles que la version du modèle de règle utilisée lors de la définition de la logique de règle. Ces informations système apparaissent dans les propriétés pour permettre les requêtes sur ces zones. Les propriétés système commencent par un préfixe IBMSystem et sont en lecture seule dans le groupe de règles métier et les classes de propriétés. Les propriétés système peuvent être ajoutées, modifiées ou supprimées. Voici un exemple de propriété système :

Nom de la propriété	**Valeur de la propriété**
IBMSystemVersion	6.2.0

Remarque : les valeurs du nom, de l'espace de nom et du nom affiché d'un groupe de règles métier sont traitées en tant que propriétés système dans le cadre de requêtes, et font partie de la liste de propriétés à extraire pour un groupe de règles métier à l'aide de la méthode getProperties. Toutefois, ces propriétés ne sont pas définies en tant qu'éléments de propriétés en cours dans l'artefact de groupe de règles métier et n'apparaissent pas comme propriétés dans WebSphere Integration Developer, dans la mesure où elles sont définies avec des éléments uniques et distincts dans le groupe de règles métier. Elles sont fournies uniquement pour offrir davantage d'options de requête.

Les propriétés définies par l'utilisateur peuvent être utilisées pour détenir des informations spécifiques aux utilisateurs, ainsi que dans les requêtes relatives aux groupes de règles métier. Ces propriétés sont disponibles en lecture-écriture.

Les propriétés d'un groupe de règles métier peuvent être extraites individuellement ou sous forme de liste (objet PropertyList). Avec l'onglet PropertyList, les méthodes de récupération des propriétés individuelles, d'ajout et de suppression des propriétés définies par l'utilisateur sont fournies.

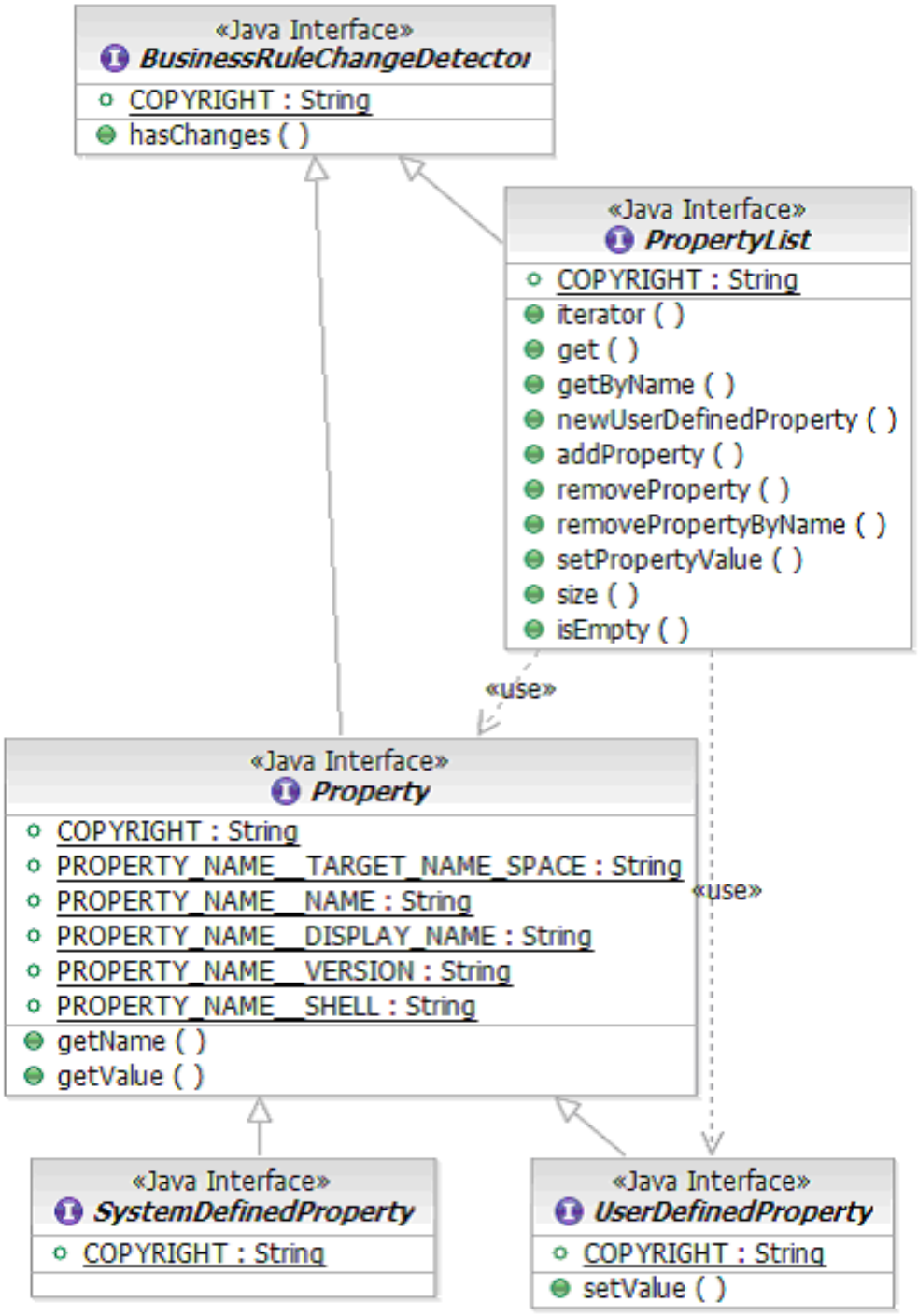


Figure 60. Diagramme de classes de Property et classes associées

Opération

Les opérations représentent le point de départ d'accès aux ensembles de règles et aux tables de décisions à modifier. Les opérations d'un groupe de règles métier correspondent aux opérations répertoriées dans le langage WSDL associé au composant de groupes de règles métier.

Pour chaque opération, il existe différentes cibles, chacune d'entre elles constituant une règle métier (ensemble de règles ou table de décision) :

- Cible par défaut (facultatif)
- Liste des cibles planifiées par plages de date/heure (OperationSelectionRecord)
- Liste de toutes les cibles disponibles pouvant être utilisées pour cette opération

Pour chaque opération, une cible de règle métier doit être spécifiée au minimum. Cette cible peut être OperationSelectionRecord et comporter une date de début et une date de fin spécifiques, correspondant à la période d'activation planifiée de la cible. Une cible unique par défaut peut également être définie pour l'opération, puis utilisée au cours de l'exécution si aucune cible de règle métier planifiée correspondante n'est trouvée. La classe Operation fournit des méthodes d'extraction et de définition de cible de règle métier par défaut, ainsi que des méthodes d'extraction de la liste (OperationSelectionRecordList) des cibles de règles métier planifiées. Outre la cible de règle métier par défaut et les cibles de règles métier planifiées, il existe une liste de toutes les cibles de règles métier disponibles pour l'opération. Cette liste répertorie les cibles de règles métier planifiées, la cible de règle métier par défaut, ainsi que les autres ensembles de règles ou tables de décisions non planifiés pour cette opération. Un ensemble de règles ou une table de décision non planifié(e) est associé(e) à l'opération via la liste des cibles disponibles, car elle partage implicitement les informations relatives à l'opération. Toutes les cibles de règles métier doivent prendre en charge les messages entrants et sortants de leur opération. Chaque opération étant unique sur une interface donnée, les ensembles de règles et les tables de décisions d'une opération sont uniques.

Vous pouvez planifier l'activation des ensembles de règles et tables de décisions de la liste des cibles disponibles via la création d'une méthodeOperationSelectionRecord. Dans ce cas, vous devez spécifier une date de début et une date de fin pour chaque ensemble de règles ou table de décision de la liste des cibles disponibles. La date de début doit être antérieure à la date de fin. Ces dates peuvent représenter une période incluant la date du jour, ou encore une période passée ou future. La période indiquée par ces dates ne peut pas chevaucher une autre période spécifiée par OperationSelectionRecords, une fois ajoutée à OperationSelectionRecordList et publiée. Les valeurs de date de début et de date de fin sont de type java.util.Date. Les valeurs spécifiées seront considérées comme des valeurs UTC, selon la classe java.util.Date. Une fois OperationSelectionRecord terminée, elle peut être ajoutée à OperationSelectionRecordList en vue d'être planifiée avec d'autres cibles de règles métier. Il peut exister des écarts entre les périodes spécifiées par différentes méthodes OperationSelectionRecords. Lorsqu'un écart est constaté au cours de l'exécution, la cible par défaut est utilisée. Si aucune cible par défaut n'a été spécifiée, une exception est générée. Il est recommandé de toujours spécifier une cible de règle métier par défaut.

Une cible de règle métier par défaut peut être supprimée de la liste des cibles planifiées, via la suppression de la méthodeOperationSelectionRecord de OperationSelectionRecordList. Si vous supprimez un élément OperationSelectionRecord, cela ne supprime pas la cible de règle métier correspondante de la liste des cibles de règles métier disponibles, et cela ne supprime pas non plus les autres éléments OperationSelectionRecords portant la même cible de règle métier planifiée.

Outre l'extraction d'un ensemble de règles ou d'une table de décision via la méthode OperationSelectionRecordList ou via la liste des cibles disponibles, la

classe Operation permet également d'extraire les cibles de règles métier par nom et par valeur de propriété d'espace de nom cible. Grâce aux méthodes de la classe Operation, les ensembles de règles et tables de décisions qui figurent parmi les cibles disponibles pour cette opération peuvent faire l'objet d'une requête. Les ensembles de règles et tables de décisions susceptibles de porter des valeurs de nom et d'espace de nom cible correspondantes, mais qui font partie des listes des cibles disponibles d'autres opérations ne sont pas inclus dans l'ensemble de résultats. Les méthodes getBusinessRulesByName, getBusinessRulesByTNS et getBusinessRulesByTNSAndName sont fournies pour simplifier l'extraction d'ensembles de règles et de tables de décisions spécifiques.

La classe Operation fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction du nom de l'opération
- Extraction de la description de l'opération
- Extraction et définition de la cible de règle métier par défaut
- Extraction des cibles de règles métier planifiées (OperationSelectionRecordList)
- Extraction de la liste de toutes les cibles de règles métier disponibles
- Extraction d'un ensemble de règles ou d'une table de décision de la liste des cibles disponibles, par nom ou par espace de nom cible
- Extraction du groupe de règles métier associé à l'opération

La classe OperationSelectionRecordList fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction d'un élément de la classe OperationSelectionRecord par valeur d'index
- Suppression d'un élément spécifique de la classe OperationSelectionRecord par valeur d'index
- Ajout d'un nouvel élément de la classe OperationSelectionRecord à la liste

La classe OperationSelectionRecord fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction et définition de la date de début
- Extraction et définition de la date de fin
- Extraction et définition de la cible de règle métier
- Extraction de l'opération à laquelle l'élément de la classe OperationSelectionRecord est associé

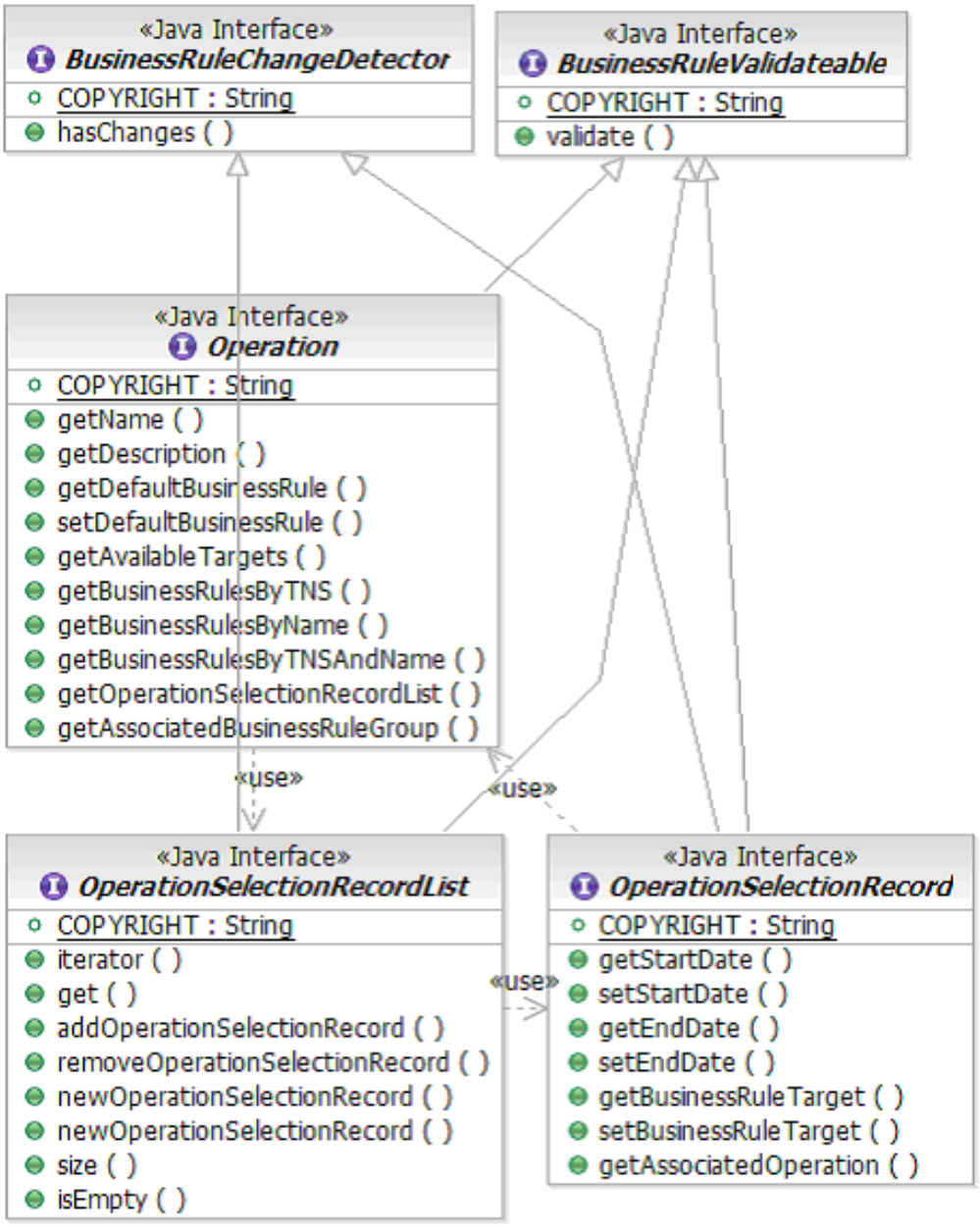


Figure 61. Diagramme de classes de Operation et classes associées

Règle métier

Les classes RuleSet et DecisionTable sont basées sur une classe générique BusinessRule et contiennent des méthodes fournissant les informations disponibles dans les ensembles de règles et les tables de décision.

A l'instar des artefacts de groupe de règles métier, les ensembles de règles et les tables de décision possèdent un nom et un espace de nom cible. La combinaison de ces valeurs doit être unique par rapport aux autres ensembles de règles et tables de décision. Par exemple, deux ensembles de règles peuvent partager la même valeur d'espace de nom cible, mais leur nom doit être différent. De même, un ensemble de règles et une table de décision peuvent porter le même nom mais ils doivent détenir des valeurs d'espace de nom cible différentes.

La copie d'une règle métier peut être réalisée à partir d'une règle métier existante lorsqu'une règle similaire doit être planifiée à une heure spécifique, avec différentes valeurs de paramètres pour les règles construites à partir de modèles. Dans la mesure où une classe de sauvegarde Java est nécessaire à l'implémentation de la règle métier, les règles ne peuvent pas être créés à partir de rien. La classe de sauvegarde Java est créée seulement au moment du déploiement. Lors de la création d'une règle, cette dernière est ajoutée à la liste des cibles disponibles pour l'opération associée à la règle d'origine. Toutefois, la règle additionnelle n'est pas conservée sauf en cas de publication du groupe de règles métier auquel l'opération est associée.

La nouvelle règle métier doit comporter un espace de nom cible ou un nom différent de la règle d'origine. Le nom affiché de la nouvelle règle métier peut rester identique à celui de la règle d'origine puisque la combinaison du nom et de l'espace de nom fournissent une valeur clé permettant d'identifier la règle métier. Dans le cadre de la règle métier, les différentes valeurs de paramètre, précédemment définies avec un modèle, peuvent être modifiées. La planification de la règle métier à une heure spécifique peut être réalisée avec OperationSelectionRecordList ou en tant que destination par défaut avec l'Opération associée à la règle métier.

La classe BusinessRule fournit des méthodes permettant de :

- Extraire l'espace de nom cible
- Extraire le nom de l'ensemble de règles ou la table de décision
- Extraire et définir le nom affiché de l'ensemble de règles ou de la table de décision
- Extraire le type de la règle métier : ensemble de règles ou table de décision
- Extraire et définir la description de la règle métier
- Extraire l'opération à laquelle la règle métier est associée.
- Créer une copie de la règle métier avec un nom et/ou un espace de nom cible différent

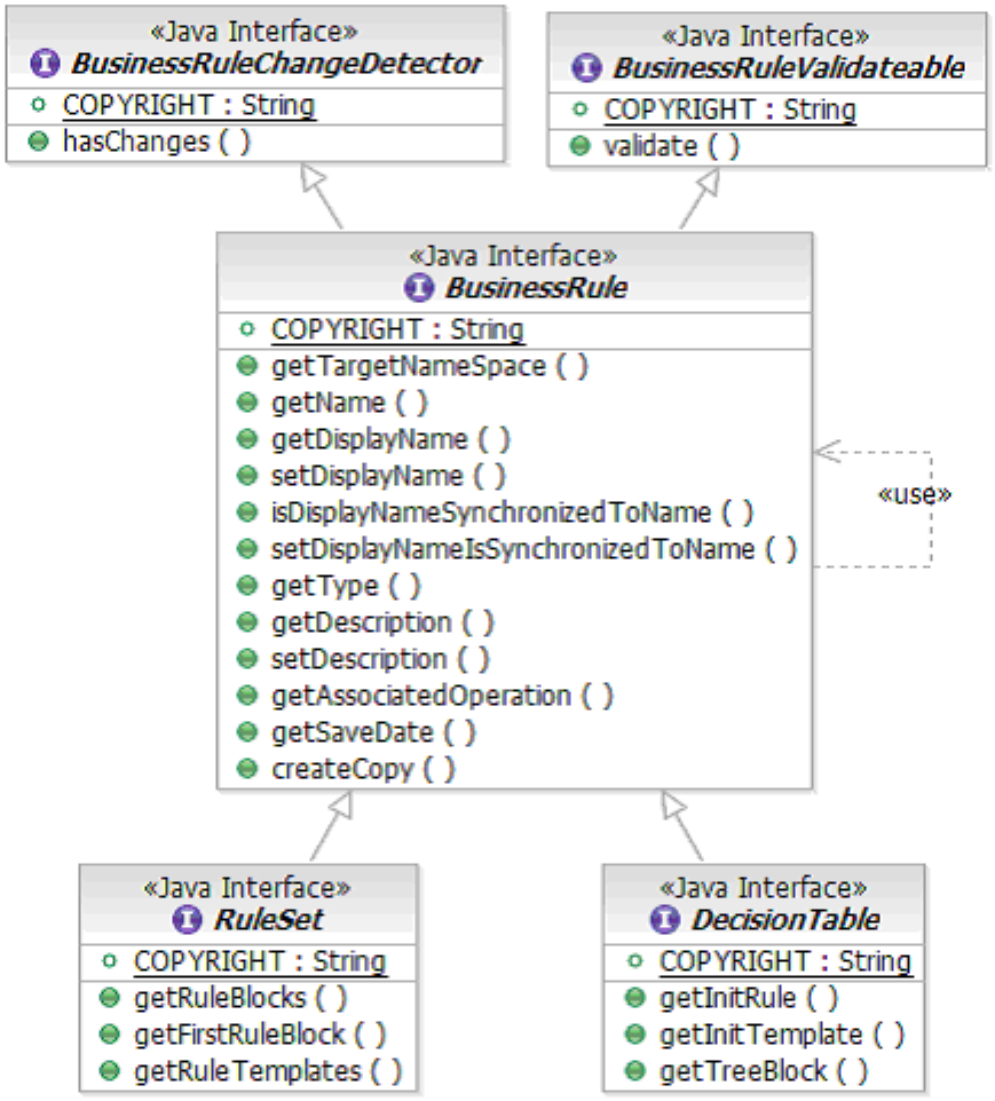


Figure 62. Diagramme de classes de BusinessRule et classes associées

Ensemble de règles

Un ensemble de règles constitue un type de règle métier. Les ensembles de règles sont généralement utilisés lorsque plusieurs règles doivent être exécutées sur la base de différentes valeurs conditionnelles. Les ensembles de règles se composent d'un bloc de règles et de modèles de règles. Le bloc de règles (`RuleBlock`) contient les différentes règles if-then et action qui composent la logique de l'ensemble de règles.

La classe `RuleSet` fournit les méthodes qui prennent en charge les aspects suivants :

- Extraction d'une liste de blocs de règles pour l'ensemble de règles
- Extraction d'une liste de modèles de règles définis dans l'ensemble de règles

A l'heure actuelle, chaque ensemble de règles ne peut contenir qu'un bloc de règles, tandis que plusieurs modèles de règles peuvent être définis dans l'ensemble

de règles. Le bloc de règles contient l'ensemble de règles qui sera exécuté lors de l'appel de l'ensemble de règles. Le bloc de règles permet de modifier l'ordre des règles. Un bloc de règles doit contenir au minimum une règle définie. Les règles (`Rule`) peuvent être définies comme des règles d'instance de modèle (TemplateInstanceRule) ou codées en dur. Si une règle if-then ou une règle action a été définie avec un modèle, elle peut être supprimée du bloc de règles. Si une nouvelle instance de règle a été créée avec un modèle, elle peut être ajoutée au bloc de règles.

Si une règle est codée en dur et qu'elle n'a pas été définie avec un modèle, elle ne peut être ni modifiée, ni supprimée du bloc de règles. Ces règles ont été conçues pour faire systématiquement partie de la logique des ensembles de règles et ne doivent pas être modifiées ou répétées au sein de cette logique.

Lorsqu'une nouvelle règle est créée avec un modèle, elle doit porter une valeur de nom unique. La liste des règles existantes peut être extraite et vérifiée avant la création de la règle.

Pour les règles codées en dur if-then et action, seuls le nom et la présentation peuvent être extraits. La présentation représente une chaîne que vous pouvez utiliser pour afficher les informations relatives à la règle dans les applications client. Pour les règles if-then ou action définies avec un modèle, vous pouvez extraire le nom et la présentation, ainsi que des informations supplémentaires. Les valeurs de paramètres spécifiques peuvent être extraites et modifiées. Si un modèle (`RuleSetRuleTemplate`) a été défini dans l'ensemble de règles, vous pouvez créer une autre instance de la règle au sein de l'ensemble de règles et définir des valeurs de paramètres. Par exemple, une règle peut indiquer qu'un client d'un niveau spécifique doit recevoir une remise d'un montant donné. Cette logique peut être définie avec un modèle de règle unique, puis répétée en modifiant les valeurs des paramètres de niveau de client (or, argent, bronze, etc.), et de montant de la remise (15 %, 10 %, 5 %, etc.).

Les paramètres d'une règle ayant été définis avec un modèle sont propres à l'instance de règle correspondante. Le modèle définit uniquement une présentation standard, ainsi que le nombre de paramètres applicables à la règle. Chaque règle définie avec un modèle peut posséder des valeurs différentes, comme l'explique l'exemple de remises appliquées à différents niveaux de clients.

La classe RuleBlock fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction d'une règle par index
- Ajout d'une règle définie avec un modèle
- Suppression d'une règle définie avec un modèle
- Modification de l'ordre établi (d'une place ou à un emplacement d'index spécifique)

La classe `RuleSetRule` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction du nom de la règle
- Extraction du nom affiché de la règle
- Extraction de la présentation de l'utilisateur
- Extraction du bloc de règle

La classe `RuleSetRuleTemplate` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Création d'une instance de modèle de règle à partir de la définition de modèle correspondante
- Extraction de l'ensemble de règles parent

La classe `TemplateInstanceRule` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction des paramètres de la règle
- Extraction de la définition de modèle qui a permis de définir la règle

La classe `Template` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction de l'ID de modèle
- Extraction du nom
- Extraction et définition du nom affiché
- Extraction et définition de la description
- Extraction des paramètres de ce modèle
- Extraction de la présentation de l'utilisateur

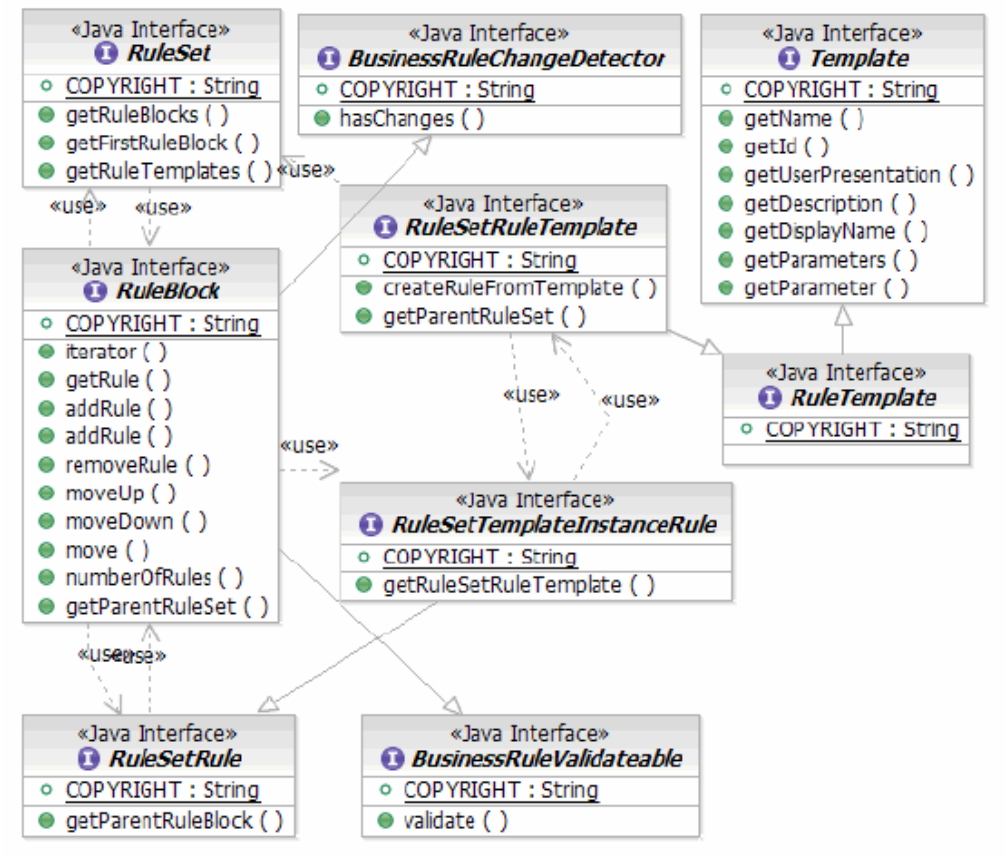


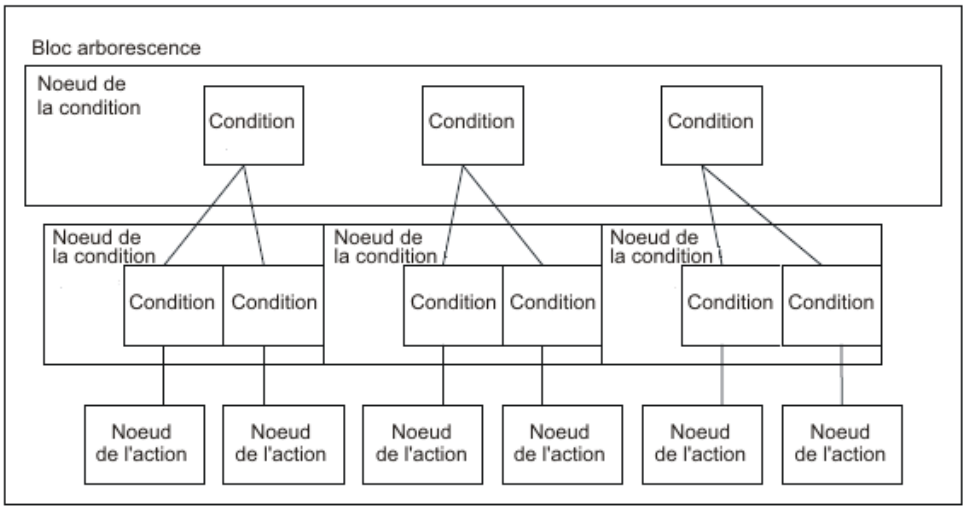
Figure 63. Diagramme de classes de BusinessRule et classes associées

table de décision

Les tables de décision représentent un autre type de règle métier que vous pouvez gérer et modifier. Elles sont généralement utilisées lorsque de nombreuses conditions doivent être évaluées et qu'un ensemble spécifique d'actions doivent être émises une fois les conditions remplies.

Les tables de décision sont semblables aux arborescences de décision, mais elles sont équilibrées. Elles comportent toujours le même nombre de conditions à évaluer et d'actions à exécuter, quelles que soient les branches résolues sur true. Une arborescence de décision peut comporter une branche ayant plus de conditions à évaluer qu'une autre.

Les tables de décision sont structurées sous la forme d'une arborescence de noeuds et sont définies par un `TreeBlock`. Différents `TreeNodes` composent le `TreeBlock`. Les `TreeNodes` peuvent être des noeuds de condition ou d'action. Les noeuds de condition sont les branches d'évaluation. A la fin des branches, les noeuds d'action contiennent les actions d'arborescence appropriées à émettre et pour lesquelles toutes les conditions doivent avoir pour résultat true. Le nombre de niveaux de noeuds de condition est illimité, mais il ne peut y avoir qu'un seul niveau de noeuds d'action.



Les tables de décision peuvent également comporter une règle d'initialisation (règle init) qui peut être émise avant vérification des conditions de la table.

La classe `DecisionTable` contient des méthodes permettant de :

- Extraire le bloc d'arborescence de noeuds d'arborescences (noeuds de condition et d'action)
- Extraire l'instance de règle init
- Extraire le modèle de règle s'il est défini

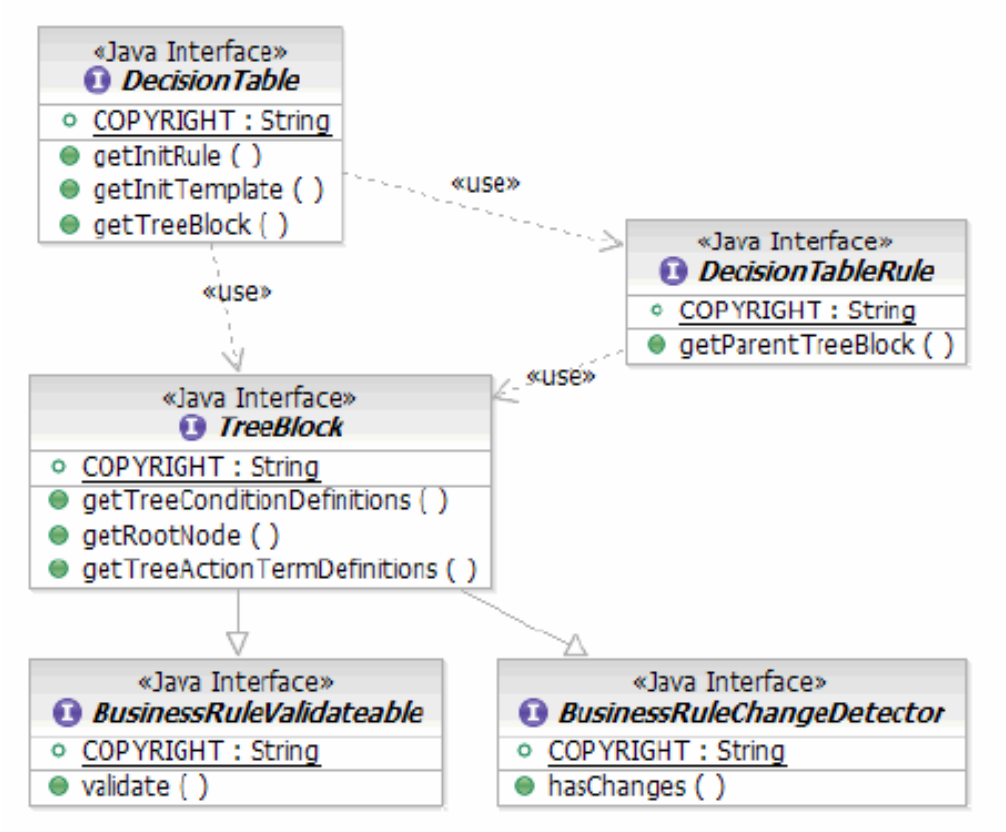
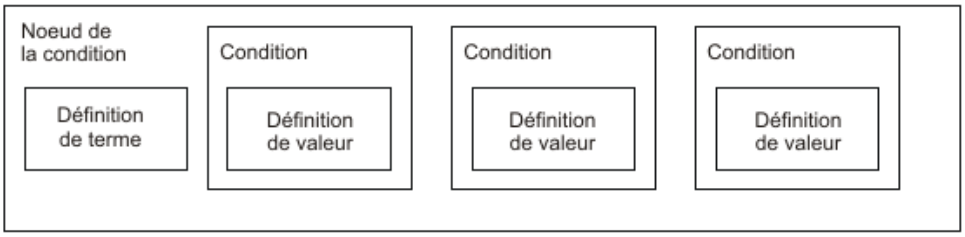


Figure 64. Diagramme de classes de DecisionTable et classes associées

Le `TreeBlock` d'une table de décision contient les différents noeuds de condition et d'action. Chaque noeud de condition (`ConditionNode`) abrite une définition de terme (`TreeConditionTermDefinition`) et de un à n arcs de cas (`CaseEdge`). La définition de terme contient l'opérande de gauche de l'expression de condition. Les arcs de cas contiennent les définitions de valeurs représentant les opérandes de droite, à utiliser dans l'expression de condition. Par exemple, dans l'expression (statut == “or”) la définition de terme est “statut” et “or” est la définition de valeur dans l'arc de cas. L'ensemble des arcs de cas d'un noeud de condition partagent la définition du terme et diffèrent uniquement par leur valeur (`TreeConditionValueDefinition`). Pour poursuivre avec cet exemple, un autre arc de cas dans le noeud de condition peut présenter la valeur “argent”. Elle est alors également utilisée dans une expression (statut == “argent”). La seule exception à ce comportement est la définition d'une clause OTHERWISE dans le noeud de condition. Avec cette clause, il n'y a aucune définition de valeur puisqu'elle est utilisée si tous les autres arcs de cas dans le noeud de condition ont pour résultat false. Bien qu'une OTHERWISE ne soit pas un arc de cas, elle possède un `TreeNode` impossible à extraire.



En ce qui concerne la définition de terme, la présentation utilisateur peut être extraite et utilisée dans les applications client. La présentation de la définition de terme est généralement une simple représentation de l'opérande de gauche (statut, dans notre exemple) et ne contient aucune marque de réservation. En ce qui concerne les arcs de cas, un modèle peut être utilisé pour définir la définition de valeur (`TreeConditionValueTemplate`). Une instance de définition de valeur de modèle (`TemplateInstanceExpression`) contient les valeurs de paramètres utilisées pour l'exécution, qui sont modifiables. Si une tentative de récupération de la définition de modèle de valeur est réalisée pour `TreeConditionValueDefinition`, non défini avec un modèle, une valeur NULL est renvoyée. Si aucun modèle n'a été utilisé pour définir la condition de valeur, une présentation utilisateur peut toujours être extraite et utilisée dans les applications client si cela a été spécifié lors de la création.

La classe `TreeBlock` contient des méthodes permettant de :

- Extraire le noeud racine de l'arborescence
- Extraire les définitions de terme de condition pour le bloc d'arborescence
- Extraire les définitions de terme d'action du bloc d'arborescence

Le noeud racine de l'arborescence est du type `TreeNode` et il représente le point de départ de la navigation dans la table de décision. La classe `TreeNode` contient des méthodes permettant de :

- Déterminer si un noeud est une clause OTHERWISE
- Extraire le noeud parent du noeud d'arborescence en cours (noeud de condition ou d'action)
- Extraire le noeud racine de l'arborescence contenant le noeud d'arborescence en cours

La classe `ConditionNode` contient des méthodes permettant de :

- Extraire les arcs de cas
- Extraire la définition de terme
- Extraire le cas OTHERWISE
- Extraire les modèles des conditions de valeur des arcs de cas pour le noeud de condition
- Ajouter au noeud une valeur de condition basée sur un modèle
- Supprimer une valeur de condition basée sur un modèle

La classe `CaseEdge` contient des méthodes permettant de :

- Extraire la liste des modèles de valeur disponibles pour la définition de valeur
- Extraire le noeud enfant (noeud de condition ou d'action)
- Extraire l'instance de la définition de modèle associée à la définition de valeur
- Extraire directement la définition de valeur sans extraire le modèle

- Définir la valeur de la définition pour utiliser une définition d'instance de modèle spécifique

La classe `TreeConditionTermDefinition` contient des méthodes permettant de :

- Extraire les modèles de définition de valeur définis pour le noeud de condition
- Extraire la présentation utilisateur du terme de condition

La classe `TreeConditionDefinition` contient des méthodes permettant de :

- Extraire la définition de terme du noeud de condition
- Extraire les définition de valeur de condition pour le noeud de condition, à partir de tous les arcs de cas
- Extraire l'orientation (ligne ou colonne)

La classe `TreeConditionValueDefinition` contient des méthodes permettant de :

- Extraire l'expression d'instance de modèle spécifique définie pour la valeur
- Extraire l'utilisateur

La classe `Template` contient des méthodes permettant de :

- Extraire l'ID système du modèle
- Extraire le nom du modèle
- Extraire les paramètres définis pour le modèle
- Extraire la présentation du modèle

La classe `TreeConditionValueTemplate` contient une méthode permettant de :

- Créer une nouvelle instance de valeur de condition de modèle

La classe `TemplateInstanceExpression` contient des méthodes permettant de :

- Extraire les paramètres de l'instance de modèle
- Extraire le modèle (`TreeConditionValueTemplate` dans le cas d'un arc de cas dans une table de décision) utilisé pour définir l'instance

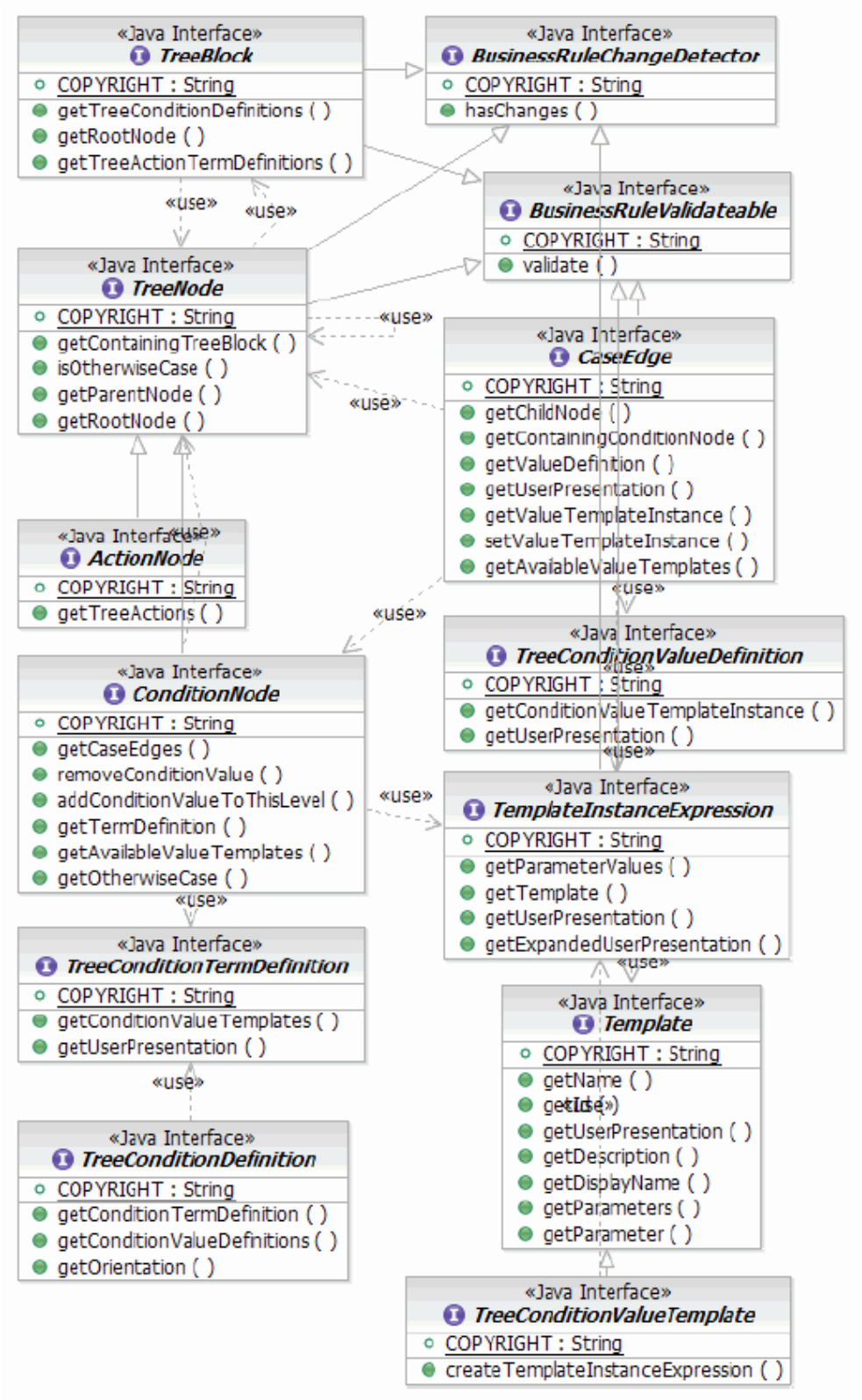


Figure 65. Diagramme de classes de TreeNode et classes associées

Lorsqu'un nouvel arc de cas est ajouté à un noeud de condition, il doit utiliser un modèle pour la définition de la valeur. Par exemple, si un nouveau cas “bronze” doit être ajouté pour la vérification de ‘statut', le modèle approprié

(TreeConditionValueTemplate) devra être utilisé pour créer un nouveau TemplateInstanceExpression, en définissant la valeur de paramètre sur “bronze”.

Lors de l'ajout d'un nouvel arc de cas, un noeud de condition enfant lui est également ajouté automatiquement. Ce noeud de condition enfant contient des arcs de cas basés sur les définitions d'arcs de cas définies pour les noeuds de condition situés au même niveau. Si des modèles ou des valeurs codées en dur sont utilisés dans les arcs de cas, ils sont ensuite également utilisés dans les arcs de cas du noeud de condition enfant. Ce noeud, ajouté automatiquement, possède également ses propres noeuds de condition enfant, créés automatiquement. Ces noeuds de condition enfant possèdent à leur tour des noeuds de condition enfant et ainsi de suite, jusqu'à ce que tous les niveaux de noeuds de condition aient été recréés.

Outre les noeuds de condition, une table de décision et plus spécifiquement un bloc d'arborescence contient également un niveau de noeuds d'action (ActionNode). Les noeuds d'action sont des noeuds terminaux qui résident à la fin de la branche de noeuds de condition et des arcs de cas. Si toutes les valeurs de condition d'une ligne d'arc de cas ont pour résultat true, un noeud d'action est atteint. Le noeud d'action possède au moins une action (TreeAction) définie. Cette action a une définition de terme et une définition de valeur. A l'instar des noeuds de condition, la définition de terme (TreeActionTermDefinition) se situe à gauche de l'expression et la définition de valeur (TemplateInstanceExpression), à droite. Par exemple, pour les différents noeuds de condition procédant à une vérification du statut, des actions peuvent définir la remise. Si la condition est (statut == “or”), l'action peut être (valeurRemise = 0.90). Pour l'action, “valeurRemise” est la définition de terme, et “= 0.90” est la définition de valeur.

La définition de terme d'une action d'arborescence est partagée avec d'autres actions d'arborescence dans d'autres noeuds d'action. Dans la mesure où chaque branche d'arcs de cas accède à une action, les mêmes définitions de terme sont utilisées. Toutefois, ces dernières peuvent différer par l'action d'arborescence et le noeud d'action. Par exemple, la valeurRemise avec le statut “or” peut être “0.90”, et “0.95” pour un statut “argent”.

Les noeuds d'action peuvent comporter plusieurs actions d'arborescence avec une définition de terme distincte et une définition de valeur distincte. Par exemple, si la remise est fixée pour un véhicule de location, outre la définition de valeurRemise, vous pouvez également affecter un niveau de véhicule spécifique. Une autre action d'arborescence peut être créée pour définir le terme “qualitéVéhicule” sur “haut de gamme” si le statut est “or”, et “valeurRemise” sur “0.90”.

La définition de valeur dans une action d'arborescence peut être créée à partir d'un modèle (TreeActionValueTemplate). La définition de modèle contient une expression (TemplateInstanceExpression) contenant des paramètres.

En plus des paramètres, la définition de valeur entière peut être modifiée par une nouvelle instance de définition de valeur, créée avec un autre modèle défini pour l'action d'arborescence.

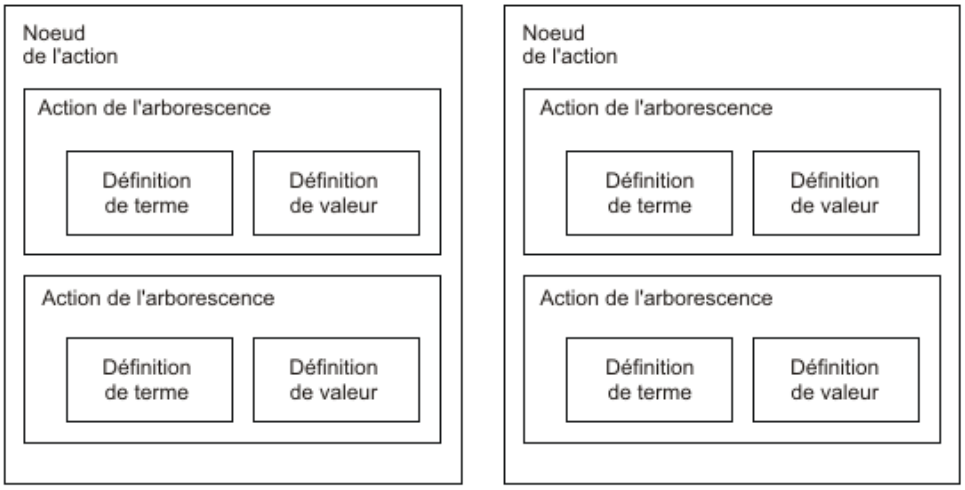
Si une définition de valeur n'est pas créée à partir d'un modèle, elle ne peut pas être modifiée. En ce qui concerne les applications client, la présentation utilisateur peut être utilisée dans l'affichage si cela a été précisé au moment de la création.

Pour les définitions de terme des actions d'arborescence, si une présentation utilisateur a été spécifiée, elle peut également être utilisée par les applications client.

Lorsqu'un nouvel arc de cas est ajouté à un noeud de condition et que différents noeuds de condition enfant sont créés, des noeuds d'action sont également créés. Contrairement aux noeuds de condition enfant et aux arcs de cas créés en fonction de la définition des arcs de cas déjà définis pour ce niveau, les noeuds d'action n'héritent pas automatiquement d'une conception existante. Seuls les marques de réservation `TreeActions` vides sont créées dans le noeud d'action. Un modèle (`TreeActionValueTemplate`) doit être utilisé pour compléter la définition d'action en créant une `TemplateInstanceExpression` pour au moins une définition de terme du noeud d'action. Avant que l'action d'arborescence soit définie avec `TemplateInstanceExpression`, elle possède des valeurs NULL spécifiées pour la valeur de présentation utilisateur et la valeur d'instance de modèle.

Lors de la création d'une nouvelle condition ayant pour résultat de nouveaux ActionNodes, les noeuds d'action sont ajoutés à droite des actions existantes pour le noeud de condition parent immédiat. Par exemple, si un statut “rubis” est ajouté à la table de décision et est censé disposer d'une remise spécifique, la condition de vérification du statut est ajoutée à droite de “or”, “argent” et “bronze”. Le noeud d'action de la remise associée à “rubis” est ajouté à droite des noeuds d'action correspondant aux arcs de cas “or”, “argent” et “bronze” .

Lors de la définition de nouvelles actions d'arborescence pour des noeuds d'action, un algorithme basé sur le noeud d'action de droite du dernier arc de cas renvoie le noeud d'action avec une action d'arborescence vide. Vous pouvez également vérifier si l'action d'arborescence possède des valeurs NULL pour la valeur de présentation utilisateur et la valeur d'instance de modèle. Une fois l'action d'arborescence obtenue, elle peut être définie avec l'instance adéquate de `TreeActionValueTemplate`.



La classe `ActionNode` contient une méthode permettant de :

- Extraire la liste des actions d'arborescences définies

La classe `TreeAction` contient des méthodes permettant de :

- Extraire la liste des modèles de valeurs disponibles, définies pour l'action d'arborescence
- Extraire la définition de terme
- Extraire l'instance de modèle de valeur définie pour l'action d'arborescence
- Extraire la présentation utilisateur de la valeur si un modèle de valeur n'a pas été utilisé
- Vérifier si l'action est un appel de service SCA (méthode `isValueNotApplicable`)
- Remplacer l'instance de modèle de valeur par une nouvelle instance

La classe `TreeActionTermDefinition` contient des méthodes permettant de :

- Extraire la présentation utilisateur pour la définition de valeur de terme
- Extraire la liste des modèles de valeurs disponibles pour l'action d'arborescence
- Vérifier si l'action est un appel de service SCA (méthode `isTermNotApplicable`)

La classe `Template` contient des méthodes permettant de :

- Extraire l'ID système du modèle
- Extraire le nom du modèle
- Extraire les paramètres définis pour le modèle
- Extraire la présentation du modèle

La classe `TreeActionValueTemplate` contient une méthode permettant de :

- Créer une nouvelle instance de modèle de valeur à partir de la définition de modèle

La classe `TemplateInstanceExpression` contient des méthodes permettant de :

- Extraire les paramètres de l'instance de modèle
- Extraire le modèle (`TreeActionValueTemplate` dans le cas d'une action d'arborescence dans une table de décision) utilisé pour définir l'instance

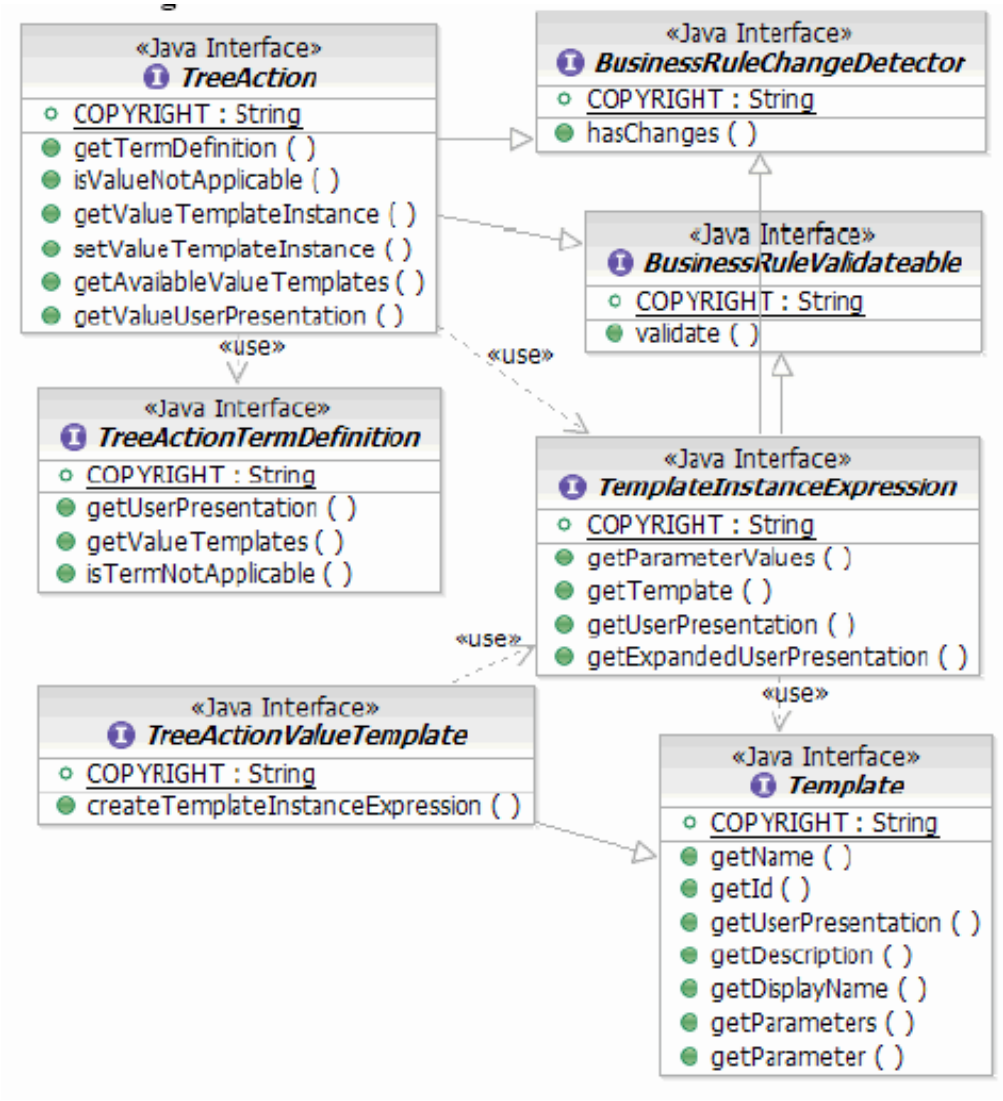


Figure 66. Diagramme de classes de TreeAction et classes associées

La définition d'une règle init pour une table de décision suit la même structure que celle d'un ensemble de règles. La règle init peut être définie avec un modèle (`DecisionTableRuleTemplate`).

Si une règle init n'a pas été créée au moment de la création, elle ne peut pas être ajoutée une fois la règle déployée.

La classe `Rule` contient des méthodes permettant de :

- Extraire le nom de la règle
- Extraire la présentation utilisateur pour la règle
- Extraire la présentation utilisateur pour la règle avec les différents paramètres définis de la règle

La classe `DecisionTableRule` contient une méthode permettant de :

- Extraire le bloc d'arborescence contenant la règle init

La classe `DecisionTableRuleTemplate` contient une méthode permettant de :

- Extraire la table de décision contenant le modèle

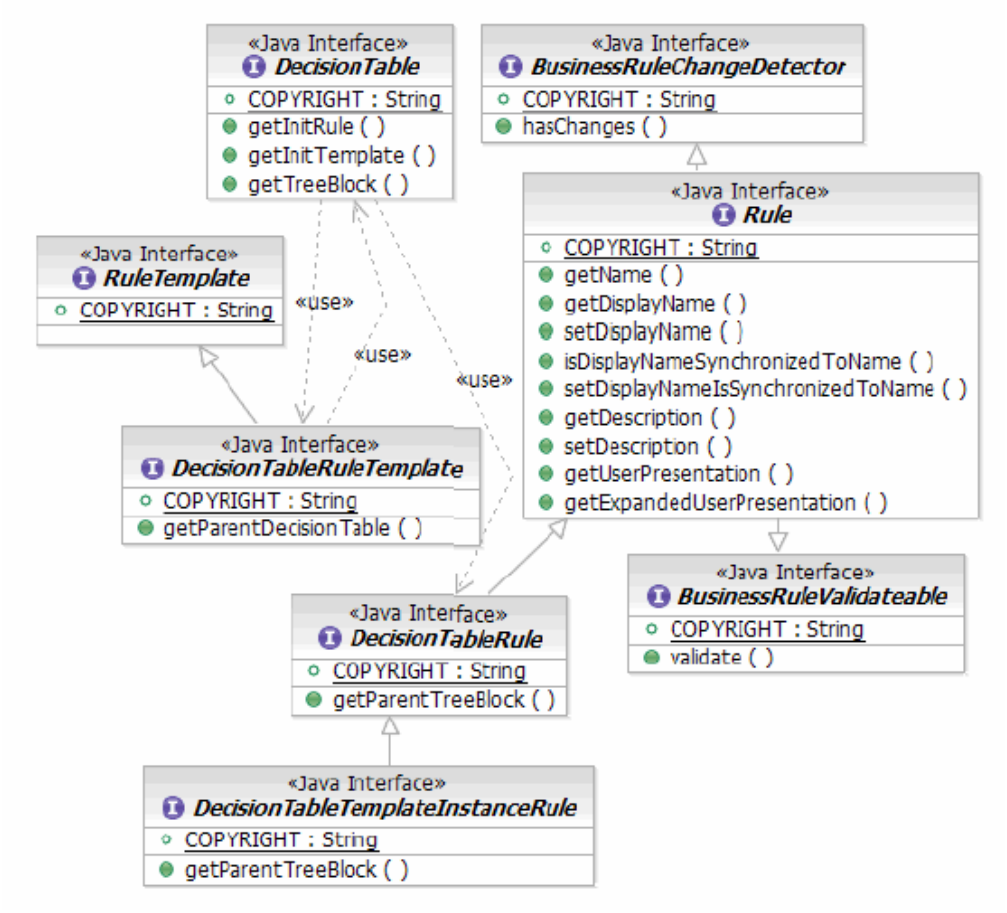


Figure 67. Diagramme de classes de DecisionTableRule et classes associées

Modèles et paramètres

Les modèles inclus dans les ensembles de règles et dans les tables de décision prennent comme base une définition commune. Les modèles possèdent des paramètres et une présentation de l'utilisateur. Les valeurs de paramètres inclus dans les modèles sont définis pour permettre d'apporter des modifications à la règle une fois que celle-ci a été déployée.

La valeur de présentation utilisateur fournit une valeur de chaîne qui peut être utilisée pour l'affichage de la règle et de différents paramètres de façon conviviale. Cette présentation sous forme de chaîne possède des marques de réservation qui permettent le remplacement des différentes valeurs de paramètres, ainsi que leur affichage. Ces marques de réservation figurent au format (<paramètre index>}. Par exemple, si la chaîne de présentation de la règle init est “Base discount is {0} %” (la remise de base s'élève à x), la marque de réservation {0} doit être remplacée par la valeur de paramètre correspondante. La chaîne de présentation ne peut pas être modifiée pour la règle ou la définition de modèle. Toutefois, les valeurs de marque de réservation peuvent être modifiées avec les valeurs de paramètre figurant dans une application client, selon la définition figurant dans le modèle. Les différents modèles incluent une méthode de simplification (`getExpandedUserPresentation`) qui renvoie une chaîne contenant toutes les valeurs de paramètre, correctement placées dans la chaîne.

Toutes les valeurs de paramètres possèdent un type de données spécifique ; toutefois, lors de l'extraction et de la définition d'une valeur de paramètre, un objet string est utilisé. La valeur de paramètre peut être considérée comme une chaîne lors du remplacement de la valeur dans la présentation utilisateur, ou encore lors de l'affectation d'une nouvelle valeur au paramètre. Le paramètre est converti dans le type de données approprié au moment de l'exécution, afin d'émettre la règle correctement. Au cours de la validation, la valeur de paramètre est comparée au type de données afin de vérifier qu'il est correct. Par exemple, si un paramètre est de type `boolean` et porte la valeur “T”, la validation ne reconnaît pas cette valeur et renvoie un message d'erreur.

Dans la définition de modèle, les valeurs de paramètres peuvent être limitées par des contraintes. Ces contraintes peuvent être définies sous forme de plage ou d'énumération. Les contraintes du paramètre seront mises en oeuvre une fois la règle validée. Si aucun modèle n'a été utilisé pour définir la valeur, seule une présentation utilisateur sera disponible. Une définition de valeur ne peut pas utiliser à la fois un modèle et une présentation utilisateur. En cas d'utilisation d'un modèle, la présentation issue de la définition de modèle est la seule présentation disponible.

La classe `Template` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction de l'ID de modèle
- Extraction du nom
- Extraction des paramètres
- Extraction de la présentation de l'utilisateur

La classe `Parameter` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction du nom du paramètre
- Extraction du type de données du paramètre
- Extraction de la contrainte associée au paramètre
- Extraction du modèle définissant le paramètre
- Création d'une valeur de paramètre

La classe `ParameterValue` fournit les méthodes qui permettent d'effectuer les opérations suivantes :

- Extraction du nom du paramètre
- Extraction de la valeur du paramètre
- Définition de la valeur du paramètre

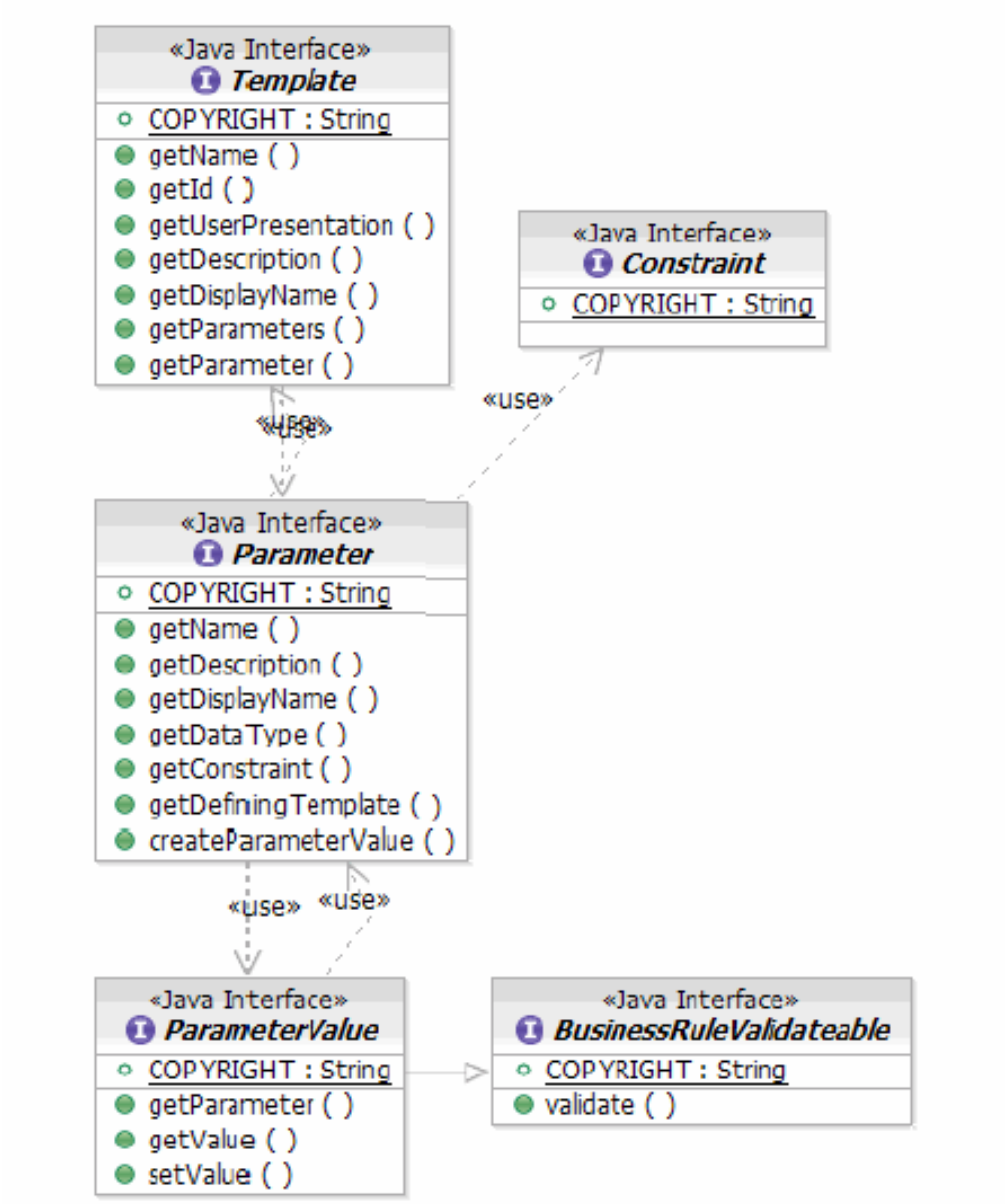


Figure 68. Diagramme de classes de Template et de Parameter, et classes associées

Validation

Parmi les objets principaux, nombreux sont ceux qui possèdent une méthode de validation ; elle permet de vérifier si les artefacts sont corrects et complets avant leur publication.

La validation qui a lieu au moment de l'apport de modifications via les classes d'API ne représente qu'un sous-ensemble de la validation globale effectuée lors de l'exécution de la commande serviceDeploy ou au moment de l'édition des artefacts dansWebSphere Integration Developer. Cela est dû aux contraintes déjà associées au groupe de règles métier (limitation des aspects modifiables au moment de l'exécution). L'utilisateur des classes peut valider la table de sélection des groupes de règles métier, la table des ensembles de règles ou la table de décision chaque fois que nécessaire (le composant de groupes de règles lui-même n'est pas

modifiable au moment de l'exécution). Lorsqu'un groupe de règles métier est publié, la table de sélection de groupes de règles, la table d'ensembles de règles et la table de décision sont validées avant leur publication dans le référentiel.

Si les artefacts sont incorrects, une exception ValidationException est générée, accompagnée de la liste des problèmes de validation rencontrés. Les différents problèmes de validation rencontrés sont documentés dans la section consacrée au traitement des exceptions.

Suivi des modifications

Pour tous les objets, vous pouvez utiliser une méthode `hasChanges` afin de vérifier si des modifications ont été apportées à l'objet et aux objets qu'il contient.

Cette méthode peut être utilisée pour vérifier les modifications et pour publier un groupe de règles métier (uniquement si certains de ses éléments ont été modifiés).

BusinessRuleManager

La classe `BusinessRuleManager` est la principale classe d'utilisation des groupes de règles, des ensembles de règles et des tables de décision.

La classe `BusinessRuleManager` comporte des méthodes permettant d'extraire les groupes de règles métier par nom, par espace de nom cible ou par propriétés personnalisées. Elle contient également une méthode de publication des modifications apportées aux groupes de règles métier, aux ensembles de règles ou aux tables de décision.

La classe `BusinessRuleManager` contient des méthodes permettant de :

- Extraire tous les groupes de règles métier
- Extraire les groupes de règles métier d'un espace de nom cible spécifique
- Extraire les groupes de règles métier d'un nom spécifique
- Extraire les groupes de règles métier d'un espace de nom cible et d'un nom spécifiques
- Extraire les groupes de règles métier contenant une propriété spécifique
- Extraire les groupes de règles métier contenant des propriétés spécifiques
- Publier des groupes de règles métier

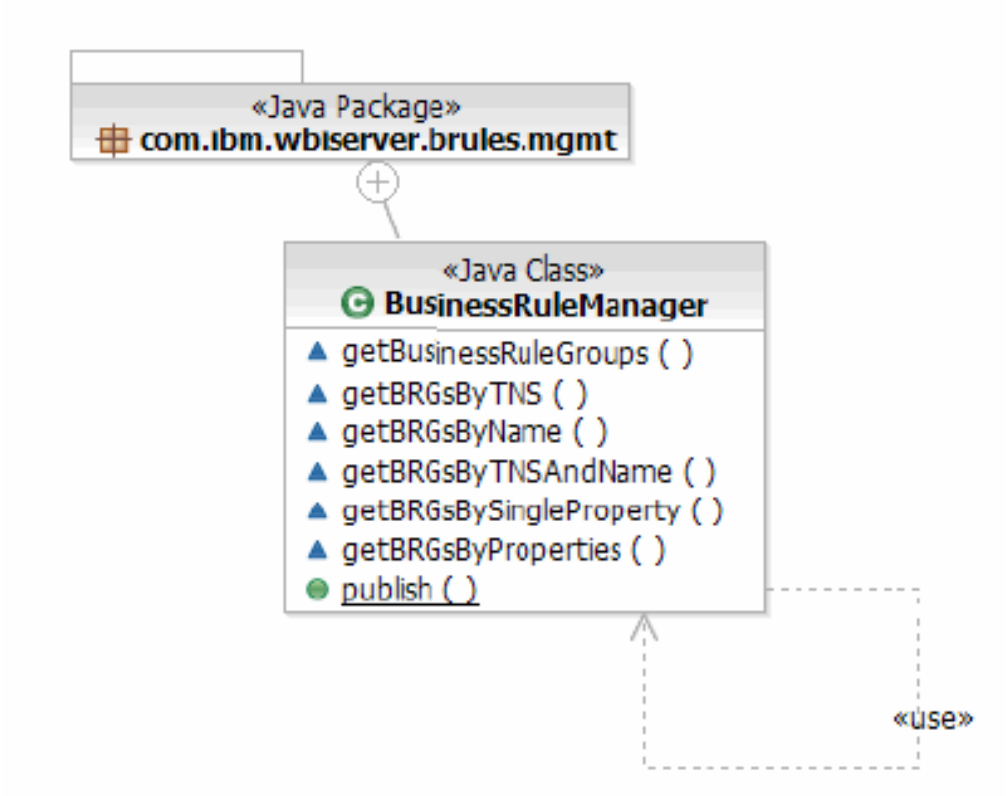


Figure 69. Diagramme de classes de BusinessRuleManager et module

Requête du composant de groupe de règles

Le composant de groupe de règles peut posséder des propriétés définies par l'utilisateur (paires nom/valeur) permettant d'affiner la liste des groupes de règles métier renvoyés par la classe. Les zones utilisables dans la requête et dans toute combinaison sont les suivantes :

- Espace de nom cible du composant de groupe de règles métier
- Nom du composant de groupe de règles métier
- Nom de propriété
- Valeur de propriété

Chaque nom de propriété ne peut être défini qu'une seule fois pour chaque composant de groupe de règles métier.

La fonction de requête prise en charge par cette classe représente un petit sous-ensemble du langage SQL entier. L'utilisateur ne fournit pas l'instruction SQL mais les valeurs des paramètres d'une propriété unique ou d'une structure arborescente contenant les informations liées à une requête à propriétés multiples sous forme de noeuds. Des noeuds d'opérateur logique et des noeuds de requête de propriété implémentent l'interface `QueryNode`. Les noeuds d'opérateur logique spécifient les opérateurs booléens (AND, OR, NOT). Ces derniers sont créés via `QueryNodeFactory`. Dans le cadre de la création de ces noeuds d'opérateur logique, les opérateurs de gauche et de droite doivent être spécifiés avec des classes `QueryNode` supplémentaires. Ces noeuds peuvent être soit un noeud de requête de propriété, soit un autre noeud d'opérateur logique. Si un noeud de requête de propriété est transmis, il contient le nom, la valeur et l'opérateur (EQUAL (==),

NOT_EQUAL (!=), LIKE ou NOTLIKE) de propriété. L'ensemble de l'interface `QueryNode` est analysée par la classe et une requête est effectuée sur les données sous-jacentes de la mémoire persistante.

Les recherches génériques sont prises en charge lors de l'utilisation des opérateurs LIKE et NOTLIKE. Les caractères '%' et '_' sont pris en charge dans les recherches génériques. Le caractère '%' est utilisé lorsqu'un nombre infini de caractères sont inconnus ou ne doivent pas être pris en compte dans la recherche. Par exemple, si une recherche doit être lancée pour tous les groupes de règles métier possédant une propriété avec un nom de Service et une valeur commençant par "Nord", la valeur doit être indiquée comme suit : "Nord%". Autre exemple, supposons que tous les services dont la valeur se termine par "Région" sont souhaités. La valeur sera alors "%Région". Le caractère '%' peut également être utilisé au milieu d'une chaîne. Par exemple, dans le cas de groupes de règles métier dont les propriétés ont les valeurs "RégionCentreNord", "RégionEstNord" et "RégionOuestNord", vous pouvez spécifier la valeur "Région%Nord".

Le caractère '_' est utilisé lorsqu'un seul caractère est inconnu ou qu'il ne doit pas être pris en compte dans la recherche. Par exemple, si une recherche porte sur tous les groupes de règles métier pour lesquels les propriétés Service ont les valeurs "Srv1Nord", "Srv2Nord", "Srv3Nord" et "Srv4Nord", la valeur "Srv_Nord" peut être spécifiée, et les 4 groupes de règles métier comportant ces propriétés sont renvoyés. Le caractère '_' peut être utilisé plusieurs fois dans une valeur de recherche, et chaque instance indique un caractère unique à ignorer. Le caractère '_' peut être utilisé au début ou à la fin d'une valeur. Par exemple, si deux caractères doivent être ignorés dans une valeur, vous pouvez utiliser deux '_' comme dans "Svc__ud".

Pour pouvoir traiter '%' et '_' en tant que caractères littéraux plutôt qu'en tant que caractères génériques, spécifiez le caractère d'échappement '\' avant '%' ou '_'. Par exemple, si le nom de propriété est "%Remise", pour pouvoir l'utiliser dans une requête, spécifiez "\%Remise". Si le caractère '\' doit être utilisé en tant que caractère littéral, un autre caractère d'échappement '\' doit être utilisé, comme dans "Commandes\\Client". Si un seul caractère '\' est placé, sans être suivi de '%', '_' ou '\', une exception IllegalArgumentException est émise.

Les caractères génériques peuvent être utilisés uniquement sur l'opérateur de gauche (valeur de propriété). Ils ne peuvent pas être utilisés dans un nom de propriété.

Au cours de recherches portant sur la valeur d'une propriété spécifique ou lors d'une recherche de valeurs, si aucune propriété n'est renvoyée, l'artefact est ignoré de la recherche. Par exemple, si parmi 3 groupes de règles métier (A, B et C), seulement deux (A et B) ont une propriété intitulée "Service" avec des valeurs différentes (respectivement "Comptabilité" et "Expédition"), et qu'une recherche est lancée sur tous les groupes de règles métier ne comportant pas de propriété "Service" définie sur "Comptabilité", seul le groupe de règles métier dont la propriété "Service" est définie, mais n'équivaut pas à "Comptabilité" (groupe de règles métier B), est renvoyé. Le groupe de règles métier (C), qui ne comporte pas de propriété "Service", n'est pas renvoyé puisque cette propriété n'est pas définie.

En cas d'utilisation de propriétés de recherche, deux propriétés spéciales intitulées *IBMSystemName* et *IBMSystemTargetNameSpace* servent aux recherches basées sur le nom et l'espace de nom d'un artefact. Ces valeurs peuvent également être extraites à l'aide des méthodes `getName` et `getTargetNameSpace`.

La classe prend en charge les méthodes de requête suivantes :

```
List getBRGsByTNS (string tNSName, Operator op, int skip, int threshold)
List getBRGByName(string Name, Operator op, int skip, int threshold)
List getBRGsByTNSAndName (string tNSName, Operator, tNSOp, string
 name, Operator nameOp, int skip, int threshold)
List getBRGsBySingleProperty (string propertyName, string propertyValue,
 Operator op, int skip, int threshold)
List getBRGsByProperties (QueryNode queryTree, int skip, int threshold)
```

Les paramètres 'skip' et 'threshold' permettent à l'utilisateur d'extraire une liste partielle de résultats jusqu'au seuil spécifié. La valeur zéro pour ces deux paramètres renvoie la liste entière de résultats. Le curseur n'est pas maintenu dans l'ensemble de résultats à partir d'un appel de requête. Si une valeur de saut est utilisée, il est possible que des ajouts ou des suppressions aient été apportés à l'ensemble de résultats et qu'une demande ultérieure renvoie alors des groupes de règles métier situés dans un précédant ensemble de résultats.

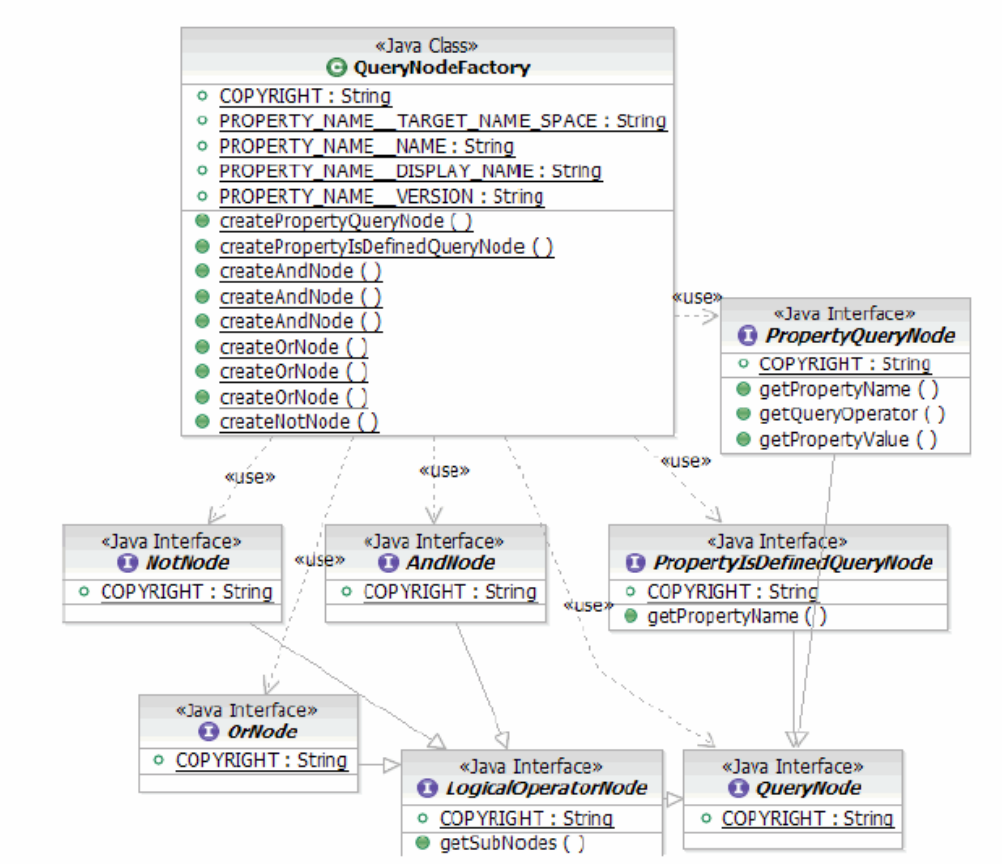


Figure 70. Diagramme de classes de QueryNodeFactory et classes associées

Les noeuds de l'arborescence permettent à l'utilisateur de spécifier une expression de recherche à l'aide des opérateurs booléens, des caractères génériques (% et échappement) et de la paire propriété/valeur. L'opérateur est valide uniquement pour les valeurs, l'opérateur de la propriété est toujours représenté par les signes égal (==).

Publication

La publication des modifications de règles métier est réalisée au niveau du composant de groupe de règles métier. L'utilisateur peut publier 1...n composants de groupe de règles métier. Avant l'exécution d'une opération de publication, une action de validation est réalisée sur le groupe de règles métier et sur les différents objets présents dans ce groupe (table de sélection d'opération, ensembles de règles, tables de décision, etc). Chaque demande de publication survient dans une transaction unique. En cas d'exception au cours de la validation ou de la publication de la base de données, la transaction est annulée et aucune modification de groupe de règles métier n'est publiée dans le référentiel. Cela permet aux modifications dépendantes les unes des autres dans un composant unique (par exemple, la table de sélection d'opération et un ensemble de règles) ou aux dépendances entre des composants de survenir au sein d'une opération atomique.

Au moment de la publication, une vérification est réalisée pour veiller à ce que les éléments à publier n'aient pas été modifiés par une autre transaction. Pour réduire les risques de conflit, la méthode de publication offre à l'utilisateur la possibilité de publier tous les artefacts, qu'ils soient modifiés ou non, ou uniquement les artefacts modifiés dans le groupe de règles métier. Le comportement par défaut instaure la publication de tous les artefacts. Si l'option est définie sur la publication de tous les artefacts et qu'une autre transaction a modifié les artefacts entre-temps, une exception `ChangeConflictException` est émise. Pour réduire le risque de conflit, spécifiez la publication des artefacts modifiés uniquement. En procédant ainsi, il est possible que deux utilisateurs apportent des modifications au référentiel pour deux artefacts différents dans un groupe de règles métier (par exemple, deux ensembles de règles), ce qui peut insérer des modifications incompatibles dans le groupe de règles métier. En raison de l'éventualité de cette situation, cette option doit être utilisée avec précaution.

Traitement des exceptions

Des exceptions peuvent être générées lors d'un appel de validation pour un artefact ou lors de sa publication. En cas d'erreur de validation, l'exception `ValidationException` est générée ; elle s'accompagne de la liste des problèmes rencontrés. Si un problème survient au cours de la publication car une autre transaction publie les mêmes artefacts, l'exception `ChangeConflictException` est générée. A chaque détection de la modification d'un artefact par une autre transaction, l'exception `ChangeConflictException` est générée.

Par ailleurs, une exception `SystemPropertyNotChangeableException` est générée en cas de tentative de modification d'une propriété qui duplique un nom de propriété système. En effet, les propriétés système ne peuvent pas être modifiées.

Une exception `ChangesNotAllowedException` est générée en cas de tentative d'exécution de l'opération set sur un artefact pendant sa publication.

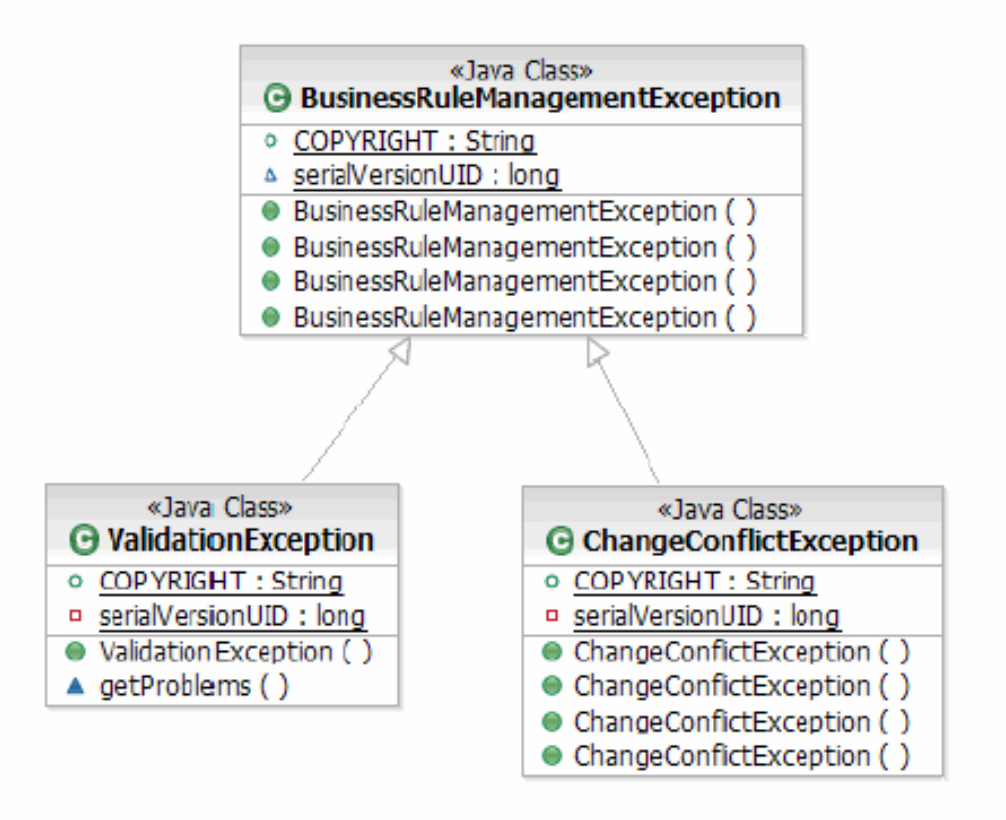


Figure 71. Diagramme de classes de BusinessRuleManagementException et classes associées

Problèmes liés aux groupes de règles métier

Ces problèmes peuvent se poser lorsqu'un groupe de règles métier est validé ou qu'une tentative de publication de ce groupe de règles métier a lieu alors qu'une partie de ce groupe est incorrecte.

Tableau 23. Problèmes liés aux groupes de règles métier

Exception	Description
ProblemBusRuleNotInAvailTargetList	Ce problème se pose lorsqu'une règle est spécifiée en tant que règle métier par défaut pour une table de sélection d'opération, mais que l'artefact correspondant à cette règle ne figure pas dans la liste des cibles disponibles pour cette opération. Pour éviter ce problème, spécifiez une règle métier valide parmi la liste des cibles disponibles pour cette opération.
ProblemDuplicatePropertyName	Ce problème se pose en cas de tentative de création d'une propriété qui représente le double d'une propriété définie par l'utilisateur pour un groupe de règles métier spécifique. Pour éviter ce problème, vous devez utiliser un nom unique de propriété.
ProblemOperationContainsNoTargets	Ce problème se pose lorsqu'une opération n'est pas associée à une destination de règle par défaut ou à un ensemble de destinations de règle planifié. Pour éviter ce problème, vous devez définir l'opération en spécifiant au minimum une destination de règle en tant que valeur par défaut ou en tant que période planifiée.

Tableau 23. Problèmes liés aux groupes de règles métier (suite)

Exception	Description
ProblemOverlappingRanges	Ce problème se pose lorsque la date de début ou la date de fin d'un enregistrement de sélection d'opération chevauche la plage correspondante d'un autre enregistrement de sélection d'opération. Ce chevauchement de plages de dates empêche la localisation de la destination de règle à appeler. Pour éviter ce problème, vous devez vérifier la date de début ou la date de fin des autres enregistrements de sélection d'opération afin de vous assurer qu'il n'existe pas de chevauchement.
ProblemStartDateAfterEndDate	Ce problème se pose lorsque la date de début d'un enregistrement de sélection d'opération est ultérieure à la date de fin de cet enregistrement. Ce problème peut se poser pour tous les enregistrements de sélection d'opération, à l'exception de l'enregistrement par défaut, qui ne possède ni date de début, ni date de fin. Pour éviter ce problème, vous devez spécifier une date de début après avoir spécifié la date de fin d'un enregistrement de sélection d'opération.
ProblemTargetBusRuleNotSet	Ce problème se pose lorsque la règle spécifiée dans un enregistrement de sélection d'opération ne figure pas dans la liste des règles cibles disponibles. Pour éviter ce problème, vous devez spécifier une règle figurant dans la liste des cibles disponibles.
ProblemTNSAndNameAlreadyInUse	Ce problème se pose lorsqu'une nouvelle règle métier est créée et qu'elle porte un nom et un espace de nom cible déjà utilisé par un ensemble de règles ou par une table de décision. Dans ce cas, un contrôle est effectué au niveau de tous les ensembles de règles et de toutes les tables de décisions associés au groupe de règles métier en cours d'utilisation, et au niveau de tous les artefacts de règles stockés dans le référentiel. Pour éviter ce problème, vous devez utiliser un autre nom ou espace de nom cible.
ProblemWrongOperationForOpSelectionRecord	Ce problème se pose lorsqu'un nouvel enregistrement de sélection d'opération est ajouté à une liste d'enregistrements de sélection d'opération et que le fonctionnement du nouvel enregistrement ne correspond pas à celui des autres enregistrements de la liste. Pour éviter ce problème, vous devez créer une nouvelle opération à l'aide de la méthode`newOperationSelectionRecord` au niveau de l'objet approprié de la liste des enregistrement de sélection d'opération.

Problèmes liés aux ensembles de règles et aux tables de décisions

Tableau 24. Problèmes liés aux ensembles de règles et aux tables de décisions

Exception	Description
ProblemInvalidBooleanValue	Ce problème se pose si un paramètre de modèle de règle d'un ensemble de règles ou une valeur d'action ou de condition d'une table de décision reçoit une valeur autre que "true" ou "false" pour un paramètre de type booléen. "T" ou "F" sont des exemples de valeurs de paramètres incorrectes. Pour éviter ce problème, utilisez les valeurs "true" ou "false" lorsque vous faites appel à un paramètre de type booléen.
ProblemParmNotDefinedInTemplate	Ce problème se pose lorsqu'une valeur est spécifiée pour un paramètre de modèle et que ce paramètre n'est pas défini dans la liste des paramètres valides pour ce modèle. Les paramètres doivent être vérifiés avant la configuration du modèle. Cela peut se produire pour les modèles RuleTemplate, TreeActionValueTemplate, ou encore TreeConditionValueTemplate.

Tableau 24. Problèmes liés aux ensembles de règles et aux tables de décisions (suite)

Exception	Description
ProblemParmValueListContainsUnexpectedValue	Ce problème se pose lorsque des paramètres valides sont transmis avec un modèle, mais que le nombre de paramètres soit trop élevé. Dans ce cas, le nombre de paramètres doit être diminué. Cela peut se produire pour les modèles RuleTemplate, TreeActionValueTemplate, ou encore TreeConditionValueTemplate.
ProblemRuleBlockContainsNoRules	Ce problème se pose lorsque toutes les règles d'un bloc d'ensemble de règles sont supprimées et qu'une tentative de validation ou de publication de cet ensemble de règles a lieu. Dans ce cas, le bloc de règles de cet ensemble doit comporter au minimum une règle.
ProblemTemplateNotAssociatedWithRuleSet	Ce problème se pose en cas de tentative d'ajout d'une règle à un ensemble de règles, alors que cette règle a été créée avec un modèle non défini au sein de cet ensemble. Pour éviter ce problème, lorsque vous créez une nouvelle règle, vous devez utiliser un modèle défini au sein de l'ensemble de règles correspondant.
ProblemRuleNameAlreadyInUse	Ce problème se pose en cas de tentative d'ajout d'une règle à un bloc d'ensemble de règles et que cette règle porte le même nom qu'une règle déjà existante au sein de ce bloc de règles. Pour éviter ce problème, vous devez vérifier les noms des règles avant l'ajout de nouvelles règles.
ProblemTemplateParameterNotSpecified	Ce problème se pose lorsqu'un paramètre est absent lors d'une mise à jour de modèle pour l'une des règles d'un ensemble de règles ou d'une valeur d'action ou de condition d'une table de décision. Pour éviter ce problème, vous devez spécifier tous les paramètres d'un modèle.
ProblemTypeConversionError	Ce problème se pose lorsqu'un paramètre de modèle ne peut pas être converti dans le type approprié ; tous les paramètres sont considérés comme des objets String, puis convertis dans le type du paramètre (boolean, byte, short, int, long, float et double). Si la chaîne de la valeur de paramètre ne peut pas être convertie dans le type spécifié pour ce paramètre, cette erreur se produit. Pour éviter ce problème, vous devez spécifier une chaîne pouvant être convertie dans le type du paramètre (boolean, byte, short, int, long, float et double).
ProblemValueViolatesParmConstraints	Ce problème se pose lorsqu'un paramètre ne se trouve pas dans l'énumération ou dans la plage de valeurs définie dans le modèle de ce paramètre. Ce problème peut concerner les paramètres limités au niveau des énumérations ou des plages (modèles de règles d'un ensemble de règles ou valeur d'action ou de condition d'une table de décision). Pour éviter ce problème, vous devez utiliser une valeur contenue dans la plage d'énumération.
ProblemInvalidActionValueTemplate	Ce problème se pose en cas de tentative de définition d'une instance de modèle dans une opération de l'arborescence, mais que le modèle correspondant n'est pas disponible pour cette opération. Pour éviter ce problème, vous devez utiliser le modèle approprié lors de la création d'une définition de valeur pour une opération de l'arborescence.
ProblemInvalidConditionValueTemplate	Ce problème se pose en cas de tentative de définition d'une instance de modèle pour une condition de cas, alors que le modèle correspondant n'est pas disponible pour ce cas. Pour éviter ce problème, utilisez le modèle approprié lors de la création d'une définition de condition pour un cas spécifique.

Tableau 24. Problèmes liés aux ensembles de règles et aux tables de décisions (suite)

Exception	Description
ProblemTreeActionIsNull	Ce problème se pose lorsqu'une valeur de condition est créée et qu'aucune action n'a pas été définie avec une instance de modèle. Dans ce cas, vous devez utiliser un modèle provenant de ActionNode, créer une nouvelle instance de modèle et la définir dans la liste `TreeActions`.

Autorisation

Les classes ne prennent en charge aucun niveau d'autorisation. L'application client utilisant les classes doit ajouter sa propre méthode d'autorisation.

Exemples

Des exemples illustrent l'utilisation possible des différentes classes pour l'extraction des groupes de règles métier et pour l'apport de modifications à des ensembles de règles et à des tables de décisions. Ces exemples sont regroupés au sein d'un fichier ZIP que vous pouvez importer dans WebSphere Integration Developer pour les visualiser et les réutiliser.

Ils contiennent plusieurs projets.

- **BRMgmtExamples** – Projet de module contenant des artefacts de règles métier utilisés dans les différents exemples.
- **BRMgmt** – Projet Java dont les exemples figurent dans le package com.ibm.websphere.sample.brules.mgmt.
- **BRMgmtDriverWeb** – Projet Web avec interface pour l'exécution des exemples.

Les exemples sont également fournis sous forme de fichier EAR (`BRMgmtExamples.ear`) qui peuvent être émis après leur installation dans WebSphere Process Server. Une interface Web est fournie avec les exemples. Cette interface est volontairement simple, car les exemples concernent l'utilisation des classes pour l'extraction d'artefacts, l'apport de modifications et la publication de celles-ci. Elle n'est pas destinée à être une interface Web hautes performances. Les classes peuvent cependant être facilement utilisées pour l'élaboration d'interfaces Web robustes, ou encore dans d'autres applications Java pour la modification des règles métier.

Remarque : Vous pouvez télécharger les fichiers d'échange de projet (.zip) et EAR des exemples à partir de la page Business Rule Management Programming Guide for WebSphere Process Server V6.1.

L'application d'exemples peut être installée surWebSphere Process Server v6.1 ; la page d'index est accessible à l'adresse suivante :

`http://<nom_hôte>:<port>/BRMgmtDriverWeb/`

Par exemple : `http://localhost:9080/BRMgmtDriverWeb/`

Au fur et à mesure de l'émission des exemples, des modifications seront apportées aux artefacts des règles. Si tous les exemples sont émis, l'application devra être réinstallée afin d'afficher de nouveau les mêmes résultats pour tous les exemples.

Les exemples sont détaillés, avec des exemples de code et le résultat tel qu'il s'affiche dans un navigateur Web.

Des classes supplémentaires ont été créées pour l'exécution d'opérations courantes et pour faciliter l'affichage des informations dans l'exemple d'application Web. Pour plus d'informations sur les classes `Formatter` et `RuleArtifactUtility`, voir l'annexe.

Pour bien comprendre ces exemples, il convient d'examiner les différents artefacts contenus dans WebSphere Integration Developer.

Concepts associés

Exemple 1 : extraction et impression de l'ensemble des groupes de règles métier
Cet exemple présente l'extraction de tous les groupes de règles métier et l'impression des attributs, des propriétés et des opérations de chaque groupe de règles métier.

Exemple 2 : Extraire et afficher tous les groupes de règles métier, les jeux de règles et les tables de décision
Outre la fonction de l'exemple 1, cet exemple permet d'imprimer la table de sélection pour chaque opération, puis la destination des règles métier par défaut (jeu de règles ou table de décision) et les autres règles métier planifiées pour l'opération. Il imprime à la fois les jeu de règles et les tables de décision.

Exemple 3 : extraction de groupes de règles métier par propriétés multiples, avec l'opérateur AND
Cet exemple est également similaire à l'exemple 1, mais il permet uniquement d'extraire les groupes de règles métier possédant la propriété Department et la valeur “accounting”, ainsi que la propriété RuleType et la valeur “regulatory”.

Exemple 4 : extraction de groupes de règles métier par propriétés multiples, avec l'opérateur OR
Cet exemple est similaire à l'exemple 3 ; toutefois, il permet uniquement d'extraire les groupes de règles métier possédant la propriété Department et la valeur “accounting”, ou encore la propriété RuleType et la valeur “monetary”.

Exemple 5 : extraction de groupes de règles métier à l'aide d'une requête complexe
Cet exemple constitue une combinaison des exemples 3 et 4 ; il a pour but d'illustrer la création de requêtes plus complexes. Dans cet exemple, une recherche est effectuée à l'aide d'une requête qui associe 2 conditions de requête. La première condition de requête consiste à extraire les groupes de règles métier possédant la propriété Department et la valeur “General”, ou encore la propriété MissingProperty et la valeur “somevalue”. Cette condition de requête est ensuite associée, à l'aide d'un opérateur AND, à une condition contenant la propriété RuleType et la valeur “messages”.

Exemple 6 : mise à jour d'une propriété de groupe de règles métier et publication du groupe de règles métier
Dans cet exemple, l'une des propriétés d'un groupe de règles métier est mise à jour, puis le groupe de règles métier correspondant est publié.

Exemple 7 : mise à jour des propriétés contenues dans plusieurs groupes de règles métier et publication des groupes de règles métier correspondants.
Dans cet exemple, les propriétés de plusieurs groupes de règles métier sont mises à jour avant la publication des groupes de règles métier correspondants.

Exemple 8 : modification de la règle métier par défaut d'un groupe de règles métier
Dans cet exemple, la règle métier par défaut est remplacée par une autre règle métier faisant partie de la liste de cibles disponibles d'une opération spécifique.

Exemple 9 : planification d'une autre règle d'opération au sein d'un groupe de règles métier
Dans cet exemple, une règle métier est planifiée en vue d'être active pendant une durée d'une heure à compter de l'heure de la publication d'une opération spécifique.

Exemple 10 : modification d'une valeur de paramètre dans un modèle d'un ensemble de règles
Dans cet exemple, une instance de règle définie avec un modèle est modifiée en changeant une valeur de paramètre, puis publiée.

Exemple 11 : Ajouter une nouvelle règle depuis un modèle vers un jeu de règles
Dans cet exemple, une nouvelle règle est ajoutée à un jeu de règles, à partir d'un

modèle. Avant la création de l'instance de règle, des paramètres sont définis pour cette instance.

Exemple 12 : Modifier et publier un modèle d'une table de décision en changeant la valeur d'un paramètre
Dans cet exemple, une condition et une action (toutes deux définies avec des modèles) sont modifiées dans une table de décision, en changeant les valeurs des paramètres avant publication.

Exemple 13 : Ajout d'une valeur de condition et d'actions dans une table de décision
Dans cet exemple, une valeur de condition et une action vont être ajoutées à une table de décision. Pour ajouter une valeur de condition à une table de décision, vous pouvez utiliser un modèle.

Exemple 14 : Gestion des erreurs dans un jeu de règles
Cet exemple explique comment identifier des incidents dans un jeu de règles et déterminer la nature de l'incident, afin d'afficher le message approprié ou de mettre en oeuvre l'action nécessaire pour corriger la situation.

Exemple 15 : Gestion des erreurs dans un groupe de règles métier
Cet exemple est similaire à l'exemple 14, car il montre comment gérer les incidents qui peuvent se produire lors de la publication d'un groupe de règles métier. Il montre comment déterminer la nature de l'incident afin d'imprimer le message correspondant ou d'exécuter l'action appropriée.

Autres exemples de requêtes
Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Exemple 1 : extraction et impression de l'ensemble des groupes de règles métier

Cet exemple présente l'extraction de tous les groupes de règles métier et l'impression des attributs, des propriétés et des opérations de chaque groupe de règles métier.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;
```

En ce qui concerne les classes de gestion des règles métier, veillez à les utiliser dans le module com.ibm.wbiserver.brules.mgmt, et pas dans le module com.ibm.wbiserver.brules ou dans tout autre module. Ces autres modules concernent les classes internes IBM.

```
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import
com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;

public class Example1 {
static Formatter out = new Formatter();
static public String executeExample1()
{
try
{
out.clear();
```

La classe BusinessRuleManager est la principale classe d'extraction des groupes de règles métier et de publication des modifications des groupes de règles métier. Cela inclut l'utilisation et la modification des artefacts de règle tels que les ensembles de règles et les tables de décision. La classe BusinessRuleManager comporte de nombreuses méthodes permettant de simplifier l'extraction de groupes de règles métier spécifiques par nom, espace de nom et propriétés.

```
// Récupérer tous les groupes de règles métier
List<BusinessRuleGroup> brgList = BusinessRuleManager
  .getBusinessRuleGroups(0, 0);

Iterator<BusinessRuleGroup> iterator = brgList.iterator();

BusinessRuleGroup brg = null;
// Procéder à une itération via la liste des groupes de règles métier
while (iterator.hasNext())
{
  brg = iterator.next();
  // Sortir les attributs de chaque groupe de règles métier
  out.println1nBold("Business Rule Group");
```

Les attributs de base du groupe de règles métier peuvent être extraits et affichés.

```
out.println("Nom : " + brg.getName());
out.println("Espace de nom : " +
brg.getTargetNameSpace());
out.println("Nom affiché : " +
brg.getDisplayName());
out.println("Description : " + brg.getDescription());
out.println("Fuseau horaire de présentation : "
     + brg.getPresentationTimezone());
out.println("Date de sauvegarde : " + brg.getSaveDate());
```

Les propriétés du groupe de règles métier peuvent également être extraites et modifiées.

```
PropertyList propList = brg.getProperties();

Iterator<Property> propIterator =
propList.iterator();
Property prop = null;
// Sortir des valeurs et des noms de propriétés
while (propIterator.hasNext())
{
prop = propIterator.next();
out.println("Nom de propriété : " +
prop.getName());
out.println("Valeur de propriété : " +
prop.getValue());
}
```

Les opérations du groupe de règles métier sont également disponibles, et permettent d'extraire les artefacts de règles métier tels que les ensembles de règles et les tables de décision.

```
List<Operation> opList = brg.getOperations();

Iteration<Operation> opIterator = opList.iterator();
Operation op = null;
// Sortir les opérations du groupe de règles métier
while (opIterator.hasNext())
{
op = opIterator.next();
out.println("Opération : " + op.getName());
}
out.println("");}
} catch (BusinessRuleManagementException e)
```

```
{
e.printStackTrace();
out.println(e.getMessage());
}
return out.toString();
}
}
```

Sortie du navigateur Web pour l'exemple 1.

```
Exécution de l'exemple 1

Groupe de règles métier
Nom : ApprovalValues
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : ApprovalValues
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de propriété : IBMSystemVersion
Valeur de propriété : 6.2.0
Nom de propriété : Department
Valeur de propriété : Accounting
Nom de propriété : RuleType
Valeur de propriété : regulatory
Nom de propriété : IBMSystemTargetNameSpace
Valeur de propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de propriété : IBMSystemName
Valeur de propriété : ApprovalValues
Nom de propriété : IBMSystemDisplayName
Valeur de propriété : ApprovalValues
Opération : getApprover

Groupe de règles métier
Nom : ConfigurationValues
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : ConfigurationValues
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de propriété : IBMSystemVersion
Valeur de propriété : 6.2.0
Nom de propriété : Department
Valeur de propriété : General
Nom de propriété : RuleType
Valeur de propriété : messages
Nom de propriété : IBMSystemTargetNameSpace
Valeur de propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de propriété : IBMSystemName
Valeur de propriété : ConfigurationValues
Nom de propriété : IBMSystemDisplayName
Valeur de propriété : ConfigurationValues
Opération : getMessages

Groupe de règles métier
Nom : DiscountRules
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : DiscountRules
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de propriété : Department
Valeur de propriété : Accounting
Nom de propriété : IBMSystemVersion
Valeur de propriété : 6.2.0
Nom de propriété : RuleType
Valeur de propriété : monetary
```

```
Nom de propriété : IBMSystemTargetNameSpace
Valeur de propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de propriété : IBMSystemName
Valeur de propriété : DiscountRules
Nom de propriété : IBMSystemDisplayName
Valeur de propriété : DiscountRules
Opération : calculateOrderDiscount
Opération : calculateShippingDiscount
```

Exemple 2 : Extraire et afficher tous les groupes de règles métier, les jeux de règles et les tables de décision

Outre la fonction de l'exemple 1, cet exemple permet d'imprimer la table de sélection pour chaque opération, puis la destination des règles métier par défaut (jeu de règles ou table de décision) et les autres règles métier planifiées pour l'opération. Il imprime à la fois les jeu de règles et les tables de décision.

La majeure partie de l'exemple est identique, mais répétée à des fins d'exhaustivité.

```
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
public class Example2
{
 status Formatter out = new Formatter();
 static public String executeExample2()
 {
  try
  {
    out.clear();
```

Un groupe de règles métier spécifique est extrait par son nom pour cet exemple.

```
  // Extraction de tous les groupes de règles métier
  List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByName("DiscountRules",
    QueryOperator.EQUAL, 0, 0);

  Iterator<BusinessRuleGroup> iterator = brgList.iterator();

  BusinessRuleGroup brg = null;
  // Itération dans la liste des groupes de règles métier
  while (iterator.hasNext())
   {
   brg = iterator.next();
   // Extraction des attributs pour chaque groupe de règles métier
   out.println1nBold("Groupe de règles métier");
   out.println("Nom: " + brg.getName());
   out.println("Espace de nom: " +
    brg.getTargetNameSpace());
   out.println("Nom affiché: " +
    brg.getDisplayName());
   out.println("Description: " + brg.getDescription());
   out.println("Fuseau horaire de présentation: "
    + brg.getPresentationTimezone());
   out.println("Date d'enregistrement: " + brg.getSaveDate());
```

```
PropertyList propList = brg.getProperties();

Iterator<Property> propIterator =
propList.iterator();
Property prop = null;
// Extraction des noms et des valeurs des propriétés
while (propIterator.hasNext())
{
  prop = propIterator.next();
  out.println("Nom de la propriété: " +
  prop.getName());
  out.println("Valeur de la proprité: " +
  prop.getValue());
}
```

Pour chaque opération, une table de sélection comporte une liste des différents artefacts de règle et leurs périodes d'activité planifiées. Une règle métier par défaut peut être spécifiée pour chaque opération. Il n'est pas obligatoire de spécifier une règle métier par défaut ou d'avoir une règle métier planifiée. Toutefois, vous devez avoir au moins une règle métier par défaut ou une règle métier spécifiée. Par conséquent, il est conseillé de rechercher les valeurs `null` avant d'utiliser la règle métier par défaut, ou de vérifier la taille de la liste `OperationSelectionRecordList`.

```
List<Operation> opList = brg.getOperations();

Iterator<Operation> opIterator = opList.iterator();
Operation op = null;
out.println("");
out.printlnBold("Opérations");
// Extraction des opérations pour le groupe de règles métier
while (opIterator.hasNext())
 {
   op = opIterator.next();
   out.printBold("Opération: ");
   out.println(op.getName());

   // Extraction de la règle métier par défaut pour l'opération
   BusinessRule defaultRule =
   op.getDefaultBusinessRule();
   // Si la règle par défaut est localisée, imprimer cette règle
   // à l'aide de la méthode appropriée pour le type de règle
   if (defaultRule != null)
   {
     out.printlnBold("Destination par défaut:");
```

La règle métier par défaut est de type `RuleSet` ou `DecisionTable`, et peut être convertie dans le type approprié pour traiter l'artefact de règle.

```
     if (defaultRule instanceof RuleSet)
        out.println(RuleArtifactUtility.
        intRuleSet(defaultRule));
     else
        out.print(RuleArtifactUtility.
        tDecisionTable(defaultRule));
   }
   OperationSelectionRecordList
   opSelectionRecordList = op
     .getOperationSelectionRecordList()
     ;

   Iterator<OperationSelectionRecord>
   opSelRecordIterator = opSelectionRecordList
     .iterator();
   OperationSelectionRecord record = null;
```

L'élément OperationSelectionRecord est composé de l'artefact de règle et des périodes d'activité de cet artefact de règle.

```
      while (opSelRecordIterator.hasNext())
      {
        out.println1nBold("Destination
        planifiée:");
        record = opSelRecordIterator.next();

        out.println("Date de début: " +
        record.getStartDate()
          + " - Date de fin: " +
          record.getEndDate());
        BusinessRule ruleArtifact = record
          .getBusinessRuleTarget();

        if (ruleArtifact instanceof RuleSet)
          out.println(RuleArtifactUtility.pr
          intRuleSet(ruleArtifact));
        else
          out.print(RuleArtifactUtility.prin
          tDecisionTable(ruleArtifact));
       }
    }
  }
  out.println("");
 } catch (BusinessRuleManagementException e)
 {
 e.printStackTrace();
 out.println(e.getMessage());
 return out.toString();
 }
}
```

Exemple

Sortie du navigateur Web pour l'exemple 2.

```
Groupe de règles métier
Nom : DiscountRules
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : DiscountRules
Description : null
Fuseau horaire de présentation : LOCAL
Date d'enregistrement : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : Department
Valeur de la propriété : Accounting
Nom de la propriété : IBMSystemVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : RuleType
Valeur de la propriété : monetary
Nom de la propriété : IBMSystemTargetNameSpace
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSystemName
Valeur de la propriété : DiscountRules
Nom de la propriété : IBMSystemDisplayName
Valeur de la propriété : DiscountRules

Opérations
Opération : calculateOrderDiscount
Destination par défaut :
Jeu de règles
Nom : calculateOrderDiscount
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Règle : CopyOrder
Nom affiché : CopyOrder
Description : null
```

```
Présentation utilisateur détaillée : null
Présentation utilisateur : null
Règle : FreeGiftInitialization
Nom affiché : FreeGiftInitialization
Description : null
Présentation utilisateur détaillée :
ID produit pour la cadeau gratuit = 5001AE80 Quantité = 1 Coût =
Description 0.0 = Cadeau gratuit pour une commande avec remise
Présentation utilisateur :
ID produit pour le cadeau gratuit = {0} Quantité = {1} Coût = {2}
Description = {3}Parameter Name: param0
Valeur du paramètre : 5001AE80
Nom du paramètre : param1
Valeur du paramètre : 1
Nom du paramètre : param2
Valeur du paramètre : 0.0
Nom du paramètre : param3
Valeur du paramètre : Cadeau gratuit pour une commande avec remise
Règle : Rule1
Nom affiché : Rule1
Description : null
Présentation utilisateur détaillée :
Si le client a le statut gold, appliquer une remise de 20,0
et ajouter un cadeau gratuit
Présentation utilisateur :
Si le client a le statut {0}, appliquer une remise
de {1} et ajouter un cadeau gratuit
Nom du paramètre : param0
Valeur du paramètre : gold
Nom du paramètre : param1
Valeur du paramètre : 20.0
Règle : Rule2
Nom affiché : Rule2
Description : null
Présentation utilisateur détaillée :
Si customer.status == silver, appliquer une remise de
15,0
Présentation utilisateur :
Si customer.status == {0}, appliquer une remise de {1}
Nom du paramètre : param0
Valeur du paramètre : silver
Nom du paramètre : param1
Valeur du paramètre : 15.0
Règle : Rule3
Nom affiché : Rule3
Description : Modèle pour les clients qui n'ont pas le statut gold
Présentation utilisateur détaillée :
Si customer.status == bronze, appliquer une remise de
10,0
Présentation utilisateur :
Si customer.status == {0}, appliquer une remise de {1}
Nom du paramètre : param0
Valeur du paramètre : bronze
Nom du paramètre : param1
Valeur du paramètre : 10.0

Opération : calculateShippingDiscount
Destination par défaut :
Table de décision
Nom : calculateShippingDiscount
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules

Règle d'initialisation : Rule1
Nom affiché : Rule1
Description : null
Présentation utilisateur détaillée : null
Présentation utilisateur : null
```

Exemple 3 : extraction de groupes de règles métier par propriétés multiples, avec l'opérateur AND

Cet exemple est également similaire à l'exemple 1, mais il permet uniquement d'extraire les groupes de règles métier possédant la propriété Department et la valeur “accounting”, ainsi que la propriété RuleType et la valeur “regulatory”.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.AndNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example3
{
static Formatter out = new Formatter();
static public String executeExample3()
{
  try
  {
    out.clear();
```

Les requêtes de groupes de règles métier sont composées de noeuds de requêtes qui suivent une arborescence. Chaque noeud de requête contient un terme gauche, un terme droit et une condition. Chacun de ces termes peut représenter un autre noeud de requête. Dans cet exemple, le groupe de règles métier est extrait via la combinaison de deux valeurs de propriété.

```
   // Extrait les groupes de règles métier sur la base de deux conditions
   // Crée des noeuds PropertyQueryNodes pour chaque condition
   PropertyQueryNode propertyNode1 = QueryNodeFactory
     .createPropertyQueryNode("Department",
     QueryOperator.EQUAL,"Accounting");
   PropertyQueryNode propertyNode2 = QueryNodeFactory
     .createPropertyQueryNode("RuleType", QueryOperator.EQUAL,
     "regulatory");
   // Associe les deux noeuds PropertyQueryNodes à un noeud AND
   AndNode andNode =
   QueryNodeFactory.createAndNode(propertyNode1, propertyNode2);

   // Utilise andNode lors des recherches de groupes de règles métier
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByProperties(andNode, 0, 0);

   Iterator<BusinessRuleGroup> iterator = brgList.iterator();

   BusinessRuleGroup brg = null;
   // Effectue une itération dans la liste des groupes de règles métier
   while (iterator.hasNext())
   {
     brg = iterator.next();
     // Permet d'obtenir les attributs de sortie de chaque groupe de règles métier
     out.println1nBold("Business Rule Group");
     out.println("Name: " + brg.getName());
     out.println("Namespace: " +
     brg.getTargetNameSpace());
     out.println("Display Name: " + brg.getDisplayName());
     out.println("Description: " + brg.getDescription());
```

```
    out.println("Presentation Time zone: "
    + brg.getPresentationTimezone());
    out.println("Save Date: " + brg.getSaveDate());

    PropertyList propList = brg.getProperties();

    Iterator<Property> propIterator =
    propList.iterator();
    Property prop = null;
    // Permet d'obtenir les noms et valeurs des propriétés
    while (propIterator.hasNext())
    {
      prop = propIterator.next();
      out.println("\t Property Name: " +
      prop.getName());
      out.println("\t Property Value: " +
      prop.getValue());
    }
  }
 } catch (BusinessRuleManagementException e)
 {
 e.printStackTrace();
 out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

Exemple

Résultat de navigateur Web pour l'exemple 3.

Exécution de l'exemple 3

```
Groupe de règles métier
Nom : ApprovalValues
Espace de noms : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : ApprovalValues
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : IBMSystemVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : Department
Valeur de la propriété : Accounting
Nom de la propriété : RuleType
Valeur de la propriété : regulatory
Nom de la propriété : IBMSystemTargetNameSpace
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSystemName
Valeur de la propriété : ApprovalValues
Nom de la propriété : IBMSystemDisplayName
Valeur de la propriété : ApprovalValues
```

Exemple 4 : extraction de groupes de règles métier par propriétés multiples, avec l'opérateur OR

Cet exemple est similaire à l'exemple 3 ; toutefois, il permet uniquement d'extraire les groupes de règles métier possédant la propriété Department et la valeur “accounting”, ou encore la propriété RuleType et la valeur “monetary”.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
```

```
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.OrNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example4
{
static Formatter out = new Formatter();
static public String executeExample4()
{
  try
  {
    out.clear();
```

Différentes propriétés composent la requête et permettent de renvoyer différents groupes de règles métier.

```
    // Retrieve business rule groups based on two conditions
    // Crée des noeuds PropertyQueryNodes pour chaque condition
    PropertyQueryNode propertyNode1 = QueryNodeFactory
      .createPropertyQueryNode("Department",
      QueryOperator.EQUAL,"Accounting");
    PropertyQueryNode propertyNode2 = QueryNodeFactory
      .createPropertyQueryNode("RuleType",
      QueryOperator.EQUAL,"monetary");
    // Associe les deux noeuds PropertyQueryNodes à un noeud OR
    OrNode orNode =
    QueryNodeFactory.createOrNode(propertyNode1,
      propertyNode2);
    // Utilise orNode dans les recherches de groupes de règles métier
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByProperties(orNode, 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();

    BusinessRuleGroup brg = null;
    // Effectue une itération dans la liste des groupes de règles métier
    while (iterator.hasNext())
    {
      brg = iterator.next();
      // Permet d'obtenir les attributs de chaque groupe de règles métier
      out.println\nBold("Business Rule Group");
      out.println("Name: " + brg.getName());
      out.println("Namespace: " +
      brg.getTargetNameSpace());
      out.println("Display Name: " + brg.getDisplayName());
      out.println("Description: " + brg.getDescription());
      out.println("Presentation Time zone: "
        + brg.getPresentationTimezone());
      out.println("Save Date: " + brg.getSaveDate());

      PropertyList propList = brg.getProperties();

      Iterator<Property> propIterator =
      propList.iterator();
      Property prop = null;
      // Permet d'obtenir les noms et valeurs de propriétés
      while (propIterator.hasNext())
      {
        prop = propIterator.next();
        out.println("\t Property Name: " +
        prop.getName());
        out.println("\t Property Value: " +
```

```
        prop.getValue());
      }
      out.println("");
    }
  } catch (BusinessRuleManagementException e)
 {
   e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

Exemple

Résultat de navigateur Web pour l'exemple 4.

Exécution de l'exemple 4

```
Groupe de règles métier
Nom : ApprovalValues
Espace de noms : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : ApprovalValues
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : IBMSystemVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : Department
Valeur de la propriété : Accounting
Nom de la propriété : RuleType
Valeur de la propriété : regulatory
Nom de la propriété : IBMSystemTargetNameSpace
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSystemName
Valeur de la propriété : ApprovalValues
Nom de la propriété : IBMSystemDisplayName
Valeur de la propriété : ApprovalValues

Groupe de règles métier
Nom : DiscountRules
Espace de noms : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : DiscountRules
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : Department
Valeur de la propriété : Accounting
Nom de la propriété : IBMSystemVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : RuleType
Valeur de la propriété : monetary
Nom de la propriété : IBMSystemTargetNameSpace
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSystemName
Valeur de la propriété : DiscountRules
Nom de la propriété : IBMSystemDisplayName
Valeur de la propriété : DiscountRules
```

Exemple 5 : extraction de groupes de règles métier à l'aide d'une requête complexe

Cet exemple constitue une combinaison des exemples 3 et 4 ; il a pour but d'illustrer la création de requêtes plus complexes. Dans cet exemple, une recherche est effectuée à l'aide d'une requête qui associe 2 conditions de requête. La première condition de requête consiste à extraire les groupes de règles métier possédant la

propriété Department et la valeur “General”, ou encore la propriété MissingProperty et la valeur “somevalue”. Cette condition de requête est ensuite associée, à l'aide d'un opérateur AND, à une condition contenant la propriété RuleType et la valeur “messages”.

D'autres exemples de requêtes de groupes de règles métier figurent dans l'Annexe.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Property;
import com.ibm.wbiserver.brules.mgmt.PropertyList;
import com.ibm.wbiserver.brules.mgmt.query.AndNode;
import com.ibm.wbiserver.brules.mgmt.query.OrNode;
import com.ibm.wbiserver.brules.mgmt.query.PropertyQueryNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryNodeFactory;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example5
{
static Formatter out = new Formatter();
static public String executeExample5()
{
  try
  {
    out.clear();

    // Extrait les groupes de règles métier sur la base de trois conditions ;
    // deux de celles-ci sont combinées au sein d'un noeud OR
    // Crée des noeuds PropertyQueryNodes pour chaque condition du noeud OR
    PropertyQueryNode propertyNode1 = QueryNodeFactory
      .createPropertyQueryNode("Department",
      QueryOperator.EQUAL,"General");
    PropertyQueryNode propertyNode2 = QueryNodeFactory
      .createPropertyQueryNode("MissingProperty",
    QueryOperator.EQUAL, "SomeValue");
    // Combine les deux PropertyQueryNodes au sein d'un noeud OR
    OrNode orNode =
    QueryNodeFactory.createOrNode(propertyNode1, propertyNode2);
    // Crée le troisième noeud PropertyQueryNode
      PropertyQueryNode propertyNode3 = QueryNodeFactory
      .createPropertyQueryNode("RuleType",
    QueryOperator.EQUAL,"messages");
```

La condition de gauche est combinée à la condition de droite à l'aide d'un noeud AND. AndNode constitue la racine de l'arborescence de requête.

```
    // Combine le noeud OR avec le troisième PropertyQueryNode à l'aide de :
    AndNode AndNode andNode =
    QueryNodeFactory.createAndNode(propertyNode3, orNode);

    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByProperties(andNode, 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();

    BusinessRuleGroup brg = null;
    // Effectue une itération dans la liste des groupes de règles métier
    while (iterator.hasNext())
    {
      brg = iterator.next();
      // Permet d'obtenir les attributs de chaque groupe de règles métier
```

```
      out.println1nBold("Business Rule Group");
      out.println("Name: " + brg.getName());
      out.println("Namespace: " +
        brg.getTargetNameSpace());
      out.println("Display Name: " + brg.getDisplayName());
      out.println("Description: " + brg.getDescription());
      out.println("Presentation Time zone: "
        + brg.getPresentationTimezone());
      out.println("Save Date: " + brg.getSaveDate());
      PropertyList propList = brg.getProperties();

      Iterator<Property> propIterator =
      propList.iterator();
      Property prop = null;
      // Permet d'obtenir les noms et valeurs de propriétés
      while (propIterator.hasNext())
      {
        prop = propIterator.next();
        out.println("\t Property Name: " +
        prop.getName());
        out.println("\t Property Value: " +
        prop.getValue());
      }
    }
   }   catch (BusinessRuleManagementException e)
  {
  e.printStackTrace();
  out.println(e.getMessage());
  }
 return out.toString();
 }
}
```

Exemple

Résultat de navigateur Web pour l'exemple 5.

```
Exécution de l'exemple 5

Groupe de règles métier
Nom : ConfigurationValues
Espace de noms : http://BRSamples/com/ibm/websphere/sample/brules
Nom affiché : ConfigurationValues
Description : null
Fuseau horaire de présentation : LOCAL
Date de sauvegarde : Sun Jan 06 17:56:51 CST 2008
Nom de la propriété : IBMSystemVersion
Valeur de la propriété : 6.2.0
Nom de la propriété : Department
Valeur de la propriété : General
Nom de la propriété : RuleType
Valeur de la propriété : messages
Nom de la propriété : IBMSystemTargetNameSpace
Valeur de la propriété : http://BRSamples/com/ibm/websphere/sample/brules
Nom de la propriété : IBMSystemName
Valeur de la propriété : ConfigurationValues
Nom de la propriété : IBMSystemDisplayName
Valeur de la propriété : ConfigurationValues
```

Exemple 6 : mise à jour d'une propriété de groupe de règles métier et publication du groupe de règles métier

Dans cet exemple, l'une des propriétés d'un groupe de règles métier est mise à jour, puis le groupe de règles métier correspondant est publié.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
```

```
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.UserDefinedProperty;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example6
{
static Formatter out = new Formatter();

static public String executeExample6()
{
 try
 {
   out.clear();
   out.printlnBold("Business Rule Group before publish:");
   // Extrait les groupes de règles métier à l'aide d'une seule valeur de propriété
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsBySingleProperty("Department",
     QueryOperator.EQUAL,"General", 0, 0);

   if (brgList.size() > 0)
   {
     // Extrait le premier groupe de règles métier de la liste
     BusinessRuleGroup brg = brgList.get(0);
     // Extrait la propriété du groupe de règles métier
     UserDefinedProperty userDefinedProperty =
     (UserDefinedProperty) brg
       .getProperty("Department");

     out.println("Business Rule Group: " + brg.getName());
     out.println("Department Property value: "
     + brg.getProperty("Department").getValue());
```

La méthode getProperty renvoie une propriété par référence ; les modifications apportées à la propriété sont directement répercutées au niveau du groupe de règles métier.

```
     // Modification de la valeur de propriété du groupe de règles métier
     // Cela permet de mettre à jour la valeur de la propriété directement dans
     l'objet du groupe de règles métier
     userDefinedProperty.setValue("GeneralConfig");
     // Utilise la liste d'origine ou crée une nouvelle liste
     // de groupes de règles métier
     List<BusinessRuleGroup> publishList = new
     ArrayList<BusinessRuleGroup>();
     // Ajoute le groupe de règles métier modifié à la liste
     publishList.add(brg);
```

La classe BusinessRuleManager est utilisée pour la publication des modifications apportées à un groupe de règles métier. Pour publier ces modifications, une liste est transférée à la méthode de publication BusinessRuleManager, même si un seul élément est publié.

```
     // Publie la liste contenant le groupe de règles métier modifié
     BusinessRuleManager.publish(publishList, true);

     out.println("");

     // Extrait de nouveau le groupe de règles métier pour vérifier que les
     // modifications ont été publiées
     out.printlnBold("Business Rule Group after publish:");
     brgList = BusinessRuleManager
     .getBRGsBySingleProperty("Department",
     QueryOperator.EQUAL, "GeneralConfig", 0, 0);
```

```
     brg = brgList.get(0);

     out.println("Business Rule Group: " + brg.getName());
     // Affiche la valeur de propriété pour indiquer la modification apportée
     out.println("Department Property value: "
     + brg.getProperty("Department").getValue());
   }
 } catch (BusinessRuleManagementException e)
 {
 e.printStackTrace();
 out.println(e.getMessage());
 }
return out.toString();
}
}
```

Exemple

Résultat de navigateur Web pour l'exemple 6.

```
Exécution de l'exemple 6

Groupe de règles métier avant la publication :
Groupe de règles métier : ConfigurationValues
Valeur de la propriété Department : General

Groupe de règles métier après la publication :
Groupe de règles métier : ConfigurationValues
Valeur de la propriété Department : GeneralConfig
```

Exemple 7 : mise à jour des propriétés contenues dans plusieurs groupes de règles métier et publication des groupes de règles métier correspondants.

Dans cet exemple, les propriétés de plusieurs groupes de règles métier sont mises à jour avant la publication des groupes de règles métier correspondants.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.UserDefinedProperty;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example7
{
static Formatter out = new Formatter();

static public String executeExample7()
{
  try
  {
    out.clear();
    out.printlnBold("Business Rule Group before publish:");
    // Extrait les groupes de règles métier à l'aide d'une seule valeur de propriété
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsBySingleProperty("Department",
      QueryOperator.EQUAL, "Accounting", 0, 0);

    Iterator<BusinessRuleGroup> iterator = brgList.iterator();
```

```
    BusinessRuleGroup brg = null;

    // Utilise la liste d'origine ou crée une nouvelle liste
    // de groupes de règles métier
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Effectue une itération au sein de tous les groupes de règles métier et
    // modifie la propriété
    while (iterator.hasNext())
    {
      // Extrait la propriété du groupe de règles métier
      brg = iterator.next();

      out.println("Business Rule Group: " + brg.getName());

      // Extrait la propriété du groupe de règles métier
      UserDefinedProperty prop = (UserDefinedProperty) brg
        .getProperty("Department");
      out.println("Department Property value: "
      +
      brg.getProperty("Department").getValue())
      ;

      // Modifie la valeur de propriété dans le groupe de règles métier
      // Cela permet de mettre à jour la valeur de la propriété directement dans
      l'objet du groupe de règles métier
      prop.setValue("Finance");
```

Chaque groupe de règles métier modifié est ajouté à la liste.

```
      // Ajoute le groupe de règles métier modifié à la liste
      publishList.add(brg);
      }

      // Publie la liste contenant le groupe de règles métier
      modifié
      BusinessRuleManager.publish(publishList, true);

      out.println("");

      // Extrait de nouveau les groupes de règles métier afin de vérifier que
      // les modifications ont été publiées
      out.printlnBold("Business Rule Group after
      publish:");

      brgList = BusinessRuleManager
        .getBRGsBySingleProperty("Department",
        QueryOperator.EQUAL,
        "Finance", 0, 0);
      iterator = brgList.iterator();

      while (iterator.hasNext())
      {
        brg = iterator.next();
        out.println("Business Rule Group: " +
        brg.getName());
        out.println("Department Property value: "
         +
         brg.getProperty("Department").getVa
         lue());
      }
} catch (BusinessRuleManagementException e)
{
  e.printStackTrace();
  out.println(e.getMessage());
```

```
}
return out.toString();
}
}
```

Exemple

Résultat de navigateur Web pour l'exemple 7.

```
Exécution de l'exemple 7

Groupe de règles métier avant la publication :
Groupe de règles métier : ApprovalValues
Valeur de la propriété Department : Accounting
Groupe de règles métier : DiscountRules
Valeur de la propriété Department : Accounting

Groupe de règles métier après la publication :
Groupe de règles métier : ApprovalValues
Valeur de la propriété Department : Finance
Groupe de règles métier : DiscountRules
Valeur de la propriété Department : Finance
```

Exemple 8 : modification de la règle métier par défaut d'un groupe de règles métier

Dans cet exemple, la règle métier par défaut est remplacée par une autre règle métier faisant partie de la liste de cibles disponibles d'une opération spécifique.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example8
{
static Formatter out = new Formatter();

static public String executeExample8()
{
  try
  {
    out.clear();

  // Extrait un groupe de règles métier par espace de nom et nom cible
  List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
      "http://BRSamples/com/ibm/websphere
      /sample/brules",
      QueryOperator.EQUAL,
      "DiscountRules",
      QueryOperator.EQUAL, 0, 0);

  if (brgList.size() > 0)
  {
    out.printlnBold("Business Rule Group before publish:");
    // Extrait le premier groupe de règles métier de la liste
    // Il doit s'agir du seul groupe de règles métier de la liste, car
    // la combinaison d'espace de nom et de nom cible est unique
    BusinessRuleGroup brg = brgList.get(0);
```

```
out.print("Business Rule Group: ");
out.println(brg.getName());

// Extrait le fonctionnement du groupe de règles métier dont
// la règle métier par défaut doit être mise à jour
Operation op =
brg.getOperation("calculateShippingDiscount");
```

La règle métier par défaut est extraite avant d'être mise à jour à l'aide d'une autre règle métier faisant partie de la liste de cibles disponibles de l'opération. Les Les ensembles de règles et les tables de décisions sont spécifiques aux opérations ; seuls les artefacts de règles métier relatifs à une opération peuvent être définis en tant qu'artefacts par défaut ou être programmés à un autre moment pour cette opération.

```
// Extrait la règle métier par défaut de l'opération
BusinessRule defaultRule =
op.getDefaultBusinessRule();
out.print("Default Rule: ");
out.println(defaultRule.getName());

// Extrait la liste des règles métier disponibles pour cette
opération
List<BusinessRule> ruleList =
op.getAvailableTargets();

Iterator<BusinessRule> iterator =
ruleList.iterator();
BusinessRule rule = null;

// Recherche une règle métier différente de la règle
en cours d'utilisation
// règle métier
// par défaut
while (iterator.hasNext())
{
  rule = iterator.next();
  if
  (!defaultRule.getName().equals(rule.getName()))
  {
```

La règle métier par défaut est définie pour l'objet de l'opération. L'affectation de la valeur Null à la règle métier par défaut a pour effet de supprimer la règle métier par défaut de l'opération ; toutefois, il est recommandé de spécifier une règle métier par défaut pour chaque opération.

```
    // Définit une autre règle
    // métier par défaut
    // Cette modification concerne directement
    // l'objet de l'opération
    op.setDefaultBusinessRule(rule);
    break;
  }
}
// Utilise la liste d'origine ou crée une nouvelle liste
// de groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();
// Ajoute le groupe de règles métier modifié à la liste
publishList.add(brg);
// Publie la liste contenant le groupe de règles
métier modifié
BusinessRuleManager.publish(publishList, true);

out.println("");
```

```
    // Extrait de nouveau les groupes de règles métier, afin de vérifier que
    // les modifications ont été publiées

    out.printlnBold("Business Rule Group after publish:");
    brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
      "http://BRSamples/com/ibm/websphere/sample/brules",
      QueryOperator.EQUAL, "DiscountRules",
      QueryOperator.EQUAL, 0, 0);

    brg = brgList.get(0);
    out.println("Business Rule Group: " + brg.getName());
    op = brg.getOperation("calculateShippingDiscount");

    // Extrait la règle métier par défaut de l'opération
    defaultRule = op.getDefaultBusinessRule();
    out.print("Default Rule: ");
    out.println(defaultRule.getName());
   }
 } catch (BusinessRuleManagementException e)
 {
   e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

Exemple

Résultat de navigateur Web pour l'exemple 8.

```
Exécution de l'exemple 8

Groupe de règles métier avant la publication :
Groupe de règles métier : DiscountRules
Règle par défaut : calculateShippingDiscount

Groupe de règles métier après la publication :
Groupe de règles métier : DiscountRules
Règle par défaut : calculateShippingDiscountHoliday
```

Exemple 9 : planification d'une autre règle d'opération au sein d'un groupe de règles métier

Dans cet exemple, une règle métier est planifiée en vue d'être active pendant une durée d'une heure à compter de l'heure de la publication d'une opération spécifique.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
```

```
public class Example9 {
static Formatter out = new Formatter();

static public String executeExample9()
{
 try
 {
   out.clear();

   // Extrait un groupe de règles métier par espace de nom et nom cible
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByTNSAndName(
     "http://BRSamples/com/ibm/websphere
     /sample/brules",
     QueryOperator.EQUAL,
     "DiscountRules",
     QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0)
   {
     out.println("");
     out.printlnBold("Business Rule Group before publish:");
     // Extrait le premier groupe de règles métier de la liste
     // Il doit s'agir du seul groupe de règles métier de la liste, car
     la
     // combinaison d'espace de nom et de nom cible est unique
     BusinessRuleGroup brg = brgList.get(0);

     // Extrait le fonctionnement du groupe de règles métier dont
     // une nouvelle règle métier doit être planifiée
     Operation op =
     brg.getOperation("calculateShippingDiscount");

     printOperationSelectionRecord(op);
     // Extrait la liste des règles métier disponibles pour cette opération
     List<BusinessRule> ruleList =
     op.getAvailableTargets();

     // Extrait la première règle de la liste, qui sera planifiée
     // pour l'opération
     BusinessRule rule = ruleList.get(0);

     // Extrait la liste des règles métier planifiées
     OperationSelectionRecordList opList = op
       .getOperationSelectionRecordList();
     // Crée une date de fin pour la règle métier
     Date future = new Date();
     long futureTime = future.getTime() + 3600000;
```

Pour les nouvelles règles planifiées, il est possible de spécifier une date de début et une date de fin. Si une valeur Null est affectée pour la date de début, cela indique que la règle sera active immédiatement au moment de la publication. Si une valeur Null est affectée à la date de fin, la règle ne comportera pas de date de fin. Les chevauchements de planification ne sont pas autorisés et peuvent être contrôlés via l'appel de la méthode validate au niveau de l'opération.

```
     // Crée la nouvelle règle métier planifiée en indiquant la date
     // actuelle, ce qui signifie que cette règle deviendra immédiatement active
     // au moment de la
     // publication, ainsi que la date future.
     newOperationSelectionRecord(new Date(),
       new Date(futureTime), rule);
     // Ajoute la nouvelle règle métier planifiée à la liste des
     // règles planifiées
     opList.addOperationSelectionRecord(newRecord);
```

Validation de l'opération afin de vérifier qu'il n'existe aucun chevauchement.

```
    // Valide la liste afin de vérifier l'absence de chevauchements
    List<Problem> problems = op.validate();
    if (problems.size() == 0)
    {
      // Utilise la liste d'origine ou crée une nouvelle liste
      // de groupes de règles métier
      List<BusinessRuleGroup> publishList = new
      ArrayList<BusinessRuleGroup>();
      // Ajoute le groupe de règles métier modifié à la liste
      publishList.add(brg);
      // Publie la liste contenant le groupe de règles
      métier mis à jour
      BusinessRuleManager.publish(publishList, true);
      out.println("");

      // Extrait de nouveau les groupes de règles métier afin de
      vérifier que les
      // modifications ont été publiées
      out.printlnBold("Business Rule Group after
      publish:");

      brgList =
      BusinessRuleManager.getBRGsByTNSAndName(
        "http://BRSamples/com/ibm/websphere
        /sample/brules",
        QueryOperator.EQUAL,
        "DiscountRules",
        QueryOperator.EQUAL, 0, 0);
        brg = brgList.get(0);

        op =
        brg.getOperation("calculateShippingDiscount");

        printOperationSelectionRecord(op);
      }
      // Gère l'erreur de validation
    }
 } catch (BusinessRuleManagementException e)
 {
 e.printStackTrace();
 out.println(e.getMessage());
 }
 return out.toString();
}
/*
Méthode d'impression de l'enregistrement de sélection d'opération. La
date de début et la date de fin sont imprimées, ainsi que le nom de l'artefact
de règle correspondant à l'heure planifiée.
*/
private static void printOperationSelectionRecord(Operation op)
{
 OperationSelectionRecordList opSelectionRecordList = op
   .getOperationSelectionRecordList();
 Iterator<OperationSelectionRecord> opSelRecordIterator =
 opSelectionRecordList
   .iterator();
 OperationSelectionRecord record = null;
 while (opSelRecordIterator.hasNext())
 {
   out.printlnBold("Scheduled Destination:");
   record = opSelRecordIterator.next();
   out.println("Start Date: " + record.getStartDate()
   + " - End Date: " + record.getEndDate());
   BusinessRule ruleArtifact = record.getBusinessRuleTarget();
```

```
   out.println("Rule: " + ruleArtifact.getName());
   }
 }
}
```

Exemple

Résultat de navigateur Web pour l'exemple 9.

```
Exécution de l'exemple 9

Groupe de règles métier avant la publication :
Destination planifiée :
Date de début : Thu Dec 01 00:00:00 CST 2005 - Date de fin :
Sun Dec 25 00:00:00 CST 2005
Règle : calculateShippingDiscountHoliday

Groupe de règles métier après la publication :
Destination planifiée :
Date de début : Thu Dec 01 00:00:00 CST 2005 - Date de fin :
Sun Dec 25 00:00:00 CST 2005
Règle : calculateShippingDiscountHoliday
Destination planifiée :
Date de début : Mon Jan 07 21:08:31 CST 2008 -
Date de fin : Mon Jan 07 22:08:31 CST 2008
Règle : calculateShippingDiscount
```

Exemple 10 : modification d'une valeur de paramètre dans un modèle d'un ensemble de règles

Dans cet exemple, une instance de règle définie avec un modèle est modifiée en changeant une valeur de paramètre, puis publiée.

```
package com.ibm.websphere.sample.brules.mgmt;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;

public class Example10
{
static Formatter out = new Formatter();

static public String executeExample10()
{
  try
  {
    out.clear();

    // Extraire un groupe de règles métier par espace de nom cible et
    par nom
    List<BusinessRuleGroup> brgList = BusinessRuleManager
      .getBRGsByTNSAndName(
        "http://BRSamples/com/ibm/websphere
        /sample/brules",
        QueryOperator.EQUAL,
        "ApprovalValues",
```

```
    QueryOperator.EQUAL, 0, 0);
if (brgList.size() > 0)
{
  // Obtenir le premier groupe de règles métier depuis la liste
  // Il doit être le seul groupe de règles métier de la
  liste puisque
  // la combinaison de l'espace de nom cible et du nom est
  unique
  BusinessRuleGroup brg = brgList.get(0);
  // Obtenir l'opération du groupe de règles métier comportant
  // la règle métier à modifier puisque
  // les règles métier sont associées à une opération
  // spécifique
  Operation op = brg.getOperation("getApprover");

  // Obtenir la règle métier de l'opération qui
  sera modifiée
  List<BusinessRule> ruleList =
  op.getBusinessRulesByName(
    "getApprover", QueryOperator.EQUAL, 0,
    0);

  if (ruleList.size() > 0)
  {
    out.println("");
    out.printlnBold("Rule set before publish:");
    // Obtenir la règle à modifier. Les règles sont
    uniques par
    // espace de nom cible et par nom, mais cet
    exemple
    // comporte une seule règle métier intitulée
    "getApprover"
    RuleSet ruleSet = (RuleSet) ruleList.get(0);
    out.print(RuleArtifactUtility.printRuleSet(rule
    Set));
```

Toutes les règles d'un ensemble de règles sont dans un bloc de règles. Un seul bloc de règles est pris en charge et la méthode `getFirstRuleBlock` doit être utilisée pour extraire le bloc de règles.

```
// Un ensemble de règles comporte toutes les règles définies dans un
bloc de règles
RuleBlock ruleBlock =
ruleSet.getFirstRuleBlock();

Iterator<RuleSetRule> ruleIterator =
ruleBlock.iterator();

// Procéder à l'itération via les règles du bloc de règles
pour trouver
// l'instance de règle intitulée "LargeOrderApprover"
while (ruleIterator.hasNext())
{
  RuleSetRule rule = ruleIterator.next();
```

Si une règle n'est pas définie avec un modèle de règle, seule sa présentation Web peut être extraite. Aucune mise à jour ne peut être réalisée sur une règle non définie avec un modèle. Si le nom de la règle est inconnu, il est recommandé de vérifier si elle a été définie avec un modèle.

```
// La règle doit avoir été définie avec un
modèle
// pour pouvoir être modifiée. Vérifier
si la règle en
```

```
// cours est basée sur un modèle.
if (rule instanceof
RuleSetTemplateInstanceRule)
{
```

Utilisez l'objet TemplateInstance pour créer la règle.

```
  // Obtenir l'instance du modèle de règle
  RuleSetTemplateInstanceRule
  templateInstance =
  (RuleSetTemplateInstanceRule) rule;

  // Rechercher l'instance de règle
  correspondant
  // à la règle à modifier
  if
  (templateInstance.getName().equals(
    "LargeOrderApprover"))
  {
```

Pour l'instance de modèle, seules les valeurs de paramètre peuvent être modifiées. Les paramètres sont modifiés en extrayant `ParameterValue` et en le définissant sur la valeur appropriée. Dans la mesure où `ParameterValue` est validé par référence, la mise à jour est effectuée directement sur la règle, l'ensemble de règles et le groupe de règles métier.

```
    // Obtenir le paramètre de
    l'instance de règle
    ParameterValue parameter =
    templateInstance
      .getParameterValue("par
      am2");

    // Modifier la valeur du
    paramètre
    parameter.setValue("superviso
    r");
    break;
  }
 }
}
// Utiliser la liste d'origine ou créer une nouvelle liste de
// groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Ajouter le groupe de règles métier modifié à la liste
publishList.add(brg);

// Publier la liste avec le groupe de règles métier mis à
 jour
 BusinessRuleManager.publish(publishList, true);

out.println("");
// Extraire de nouveau les groupes de règles métier pour vérifier
que les
// modifications ont été publiées
out.println("Rule set after publish:");

brgList = BusinessRuleManager
  .getBRGsByTNSAndName(
    "http://BRSamples/com/ibm/websphere/sample/brules",
    QueryOperator.EQUAL, "ApprovalValues",
    QueryOperator.EQUAL, 0, 0);

brg = brgList.get(0);
op = brg.getOperation("getApprover");
```

```
    ruleList = op.getBusinessRulesByName(
      "getApprover", QueryOperator.EQUAL, 0,0);

    ruleSet = (RuleSet) ruleList.get(0);
    out.print(RuleArtifactUtility.printRuleSet(ruleSet));
    }
  }
  } catch (BusinessRuleManagementException e)
  {
    e.printStackTrace();
    out.println(e.getMessage());
  }
  return out.toString();
  }
}
```

Exemple

Sortie du navigateur Web pour l'exemple 10.

```
Exécution de l'exemple 10

Ensemble de règles avant la publication :
Ensemble de règles
Nom : getApprover
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Règle : LargeOrderApprover
Nom affiché : LargeOrderApprover
Description : null
Présentation utilisateur détaillée :
si le nombre d'éléments Commande est supérieur à 10 et que
la commande dépasse 5000 $, l'approbation du responsable est nécessaire
Présentation utilisateur :
si le nombre d'éléments Commande est supérieur à {0} et que
la commande dépasse {1} $, l'approbation de {2} est nécessaire
Nom de paramètre : param0
Valeur de paramètre : 10
Nom de paramètre : param1
Valeur de paramètre : 5000
Nom de paramètre : param2
Valeur de paramètre : manager
Règle : DefaultApprover
Nom affiché : DefaultApprover
Description : null
Présentation utilisateur détaillée : approver = peer
Présentation utilisateur : approver = {0}
Nom de paramètre : param0
Valeur de paramètre : peer

Ensemble de règles une fois terminé :
Ensemble de règles
Nom : getApprover
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Règle : LargeOrderApprover
Nom affiché : LargeOrderApprover
Description : null
Présentation utilisateur détaillée :
si le nombre d'éléments Commande est supérieur à 10 et que
la commande dépasse 5000 $, l'approbation du superviseur est nécessaire
Présentation utilisateur :
si le nombre d'éléments Commande est supérieur
à {0} et que la commande
dépasse {1} $, l'approbation de {2} est nécessaire
Nom de paramètre : param0
Valeur de paramètre : 10
Nom de paramètre : param1
Valeur de paramètre : 5000
```

```
Nom de paramètre : param2
Valeur de paramètre : supervisor
Règle : DefaultApprover
Nom affiché : DefaultApprover
Description : null
Présentation utilisateur détaillée : approver = peer
Présentation utilisateur : approver = {0}
Nom de paramètre : param0
Valeur de paramètre : peer
```

Exemple 11 : Ajouter une nouvelle règle depuis un modèle vers un jeu de règles

Dans cet exemple, une nouvelle règle est ajoutée à un jeu de règles, à partir d'un modèle. Avant la création de l'instance de règle, des paramètres sont définis pour cette instance.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRuleTemplate;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example11
{
static Formatter out = new Formatter();

static public String executeExample11()
{
 try
 {
  out.clear();
  // Extraction d'un groupe de règles métier par nom et espace de nom
  cible
  List<BusinessRuleGroup> brgList = BusinessRuleManager
   .getBRGsByTNSAndName(
   "http://BRSamples/com/ibm/websphere
   /sample/brules",
   QueryOperator.EQUAL,
   "ApprovalValues",
   QueryOperator.EQUAL, 0, 0);

  if (brgList.size() > 0)
  {
   // Extraction du premier groupe de règles métier de la liste
   // Cela doit être le seul groupe de règles métier de la
   liste car
   // la combinaison de nom et d'espace de nom cible est
   unique
   BusinessRuleGroup brg = brgList.get(0);
   // Extraction de l'opération du groupe de règles métier qui comporte
   // la règle métier qui sera modifiée lorsque les
   // règles métier seront associées à une opération
   // spécifique
```

```
Operation op = brg.getOperation("getApprover");

// Extraction de la règle métier pour l'opération qui
sera modifiée
List<BusinessRule> ruleList =
op.getBusinessRulesByName(
 "getApprover", QueryOperator.EQUAL, 0,0);

if (ruleList.size() > 0)
{
 out.println("");
 out.printlnBold("Jeu de règles avant publication:");
 // Extraction de la règle à modifier. Les règles sont uniques par
 // nom et espace de nom cible, mais cet exemple utilise
 // une seule règle métier appelée
 "getApprover"
 RuleSet ruleSet = (RuleSet) ruleList.get(0);
 out.print(RuleArtifactUtility.printRuleSet(rule
 Set));
```

Pour ajouter une nouvelle règle au jeu de règles, le modèle approprié doit être identifié dans le jeu de règles et une instance doit être créée à partir de ce modèle. Le modèle peut être localisé grâce à son nom.

```
// Extraction de la liste des modèles de règles
ListRuleSetRuleTemplate> ruleTemplates =
ruleSet
 .getRuleTemplates();

Iterator<RuleSetRuleTemplate> templateIterator
= ruleTemplates
 .iterator();

while (templateIterator.hasNext())
{
 RuleSetRuleTemplate template =
 templateIterator.next();

 // Localisation du modèle à utiliser pour créer une
 nouvelle règle
 if
 (template.getName().equals("Template_Larg
 eOrder"))
 {
```

Pour une instance de modèle, une liste de paramètres doit être créée.

```
 // Création d'une liste pour les paramètres
 de cette instance de règle
 // modèle
 List<ParameterValue> paramList =
 new ArrayList<ParameterValue>();

 // A partir de la définition de modèle,
 extraction d'un paramètre spécifique
 // et définition d'une valeur
 Parameter param =
 template.getParameter("param0");
 ParameterValue paramValue = param
  .createParameterValue("
  20");

 // Ajout d'un paramètre à la liste
 paramList.add(paramValue);

 // Extraction du paramètre suivant et définition
 de la valeur
```

```
param = template.getParameter("param1");
paramValue =
param.createParameterValue("7500");

// Ajout d'un paramètre à la liste
paramList.add(paramValue);

// Extraction du paramètre suivant et définition
de la valeur
param =
template.getParameter("param2");
 paramValue = param
 .createParameterValue("
 Responsable de niveau 2");

// Ajout d'un paramètre à la liste
paramList.add(paramValue);
```

A partir des paramètres créés, l'instance de modèle peut être créée.

```
// Création de l'instance de règle
modèle avec la liste de
// paramètres
RuleSetTemplateInstanceRule
 templateInstance = template
 .createRuleFromTemplate
 ("ExtraLargeOrder",
 paramList);
// Extraction du bloc de règles correspondant au jeu
 de règles
RuleBlock ruleBlock =
ruleSet.getFirstRuleBlock();
```

Une fois l'instance de modèle créée, elle peut être ajoutée au bloc de règles. Elle peut ensuite être organisée parmi les autres instances de règle modèle.

```
// Ajout de la règle de modèle au
bloc de règle
ruleBlock.addRule(templateInstance)
;

 break;
 }
}

// Utilisation de la liste d'origine ou création d'une nouvelle liste
// de groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();

// Ajout du groupe de règles métier modifié à la
liste
publishList.add(brg);

// Publication de la liste avec le groupe de règles métier
mis à jour
BusinessRuleManager.publish(publishList, true);

out.println("");

// Extraction des groupes de règles métier pour
s'assurer que
// les modifications ont été publiées
out.printlnBold("Jeu de règles après publication:");

brgList = BusinessRuleManager
.getBRGsByTNSAndName(
```

```
      "http://BRSamples/com/ibm/websphere
      /sample/brules",
      QueryOperator.EQUAL,
      "ApprovalValues",
      QueryOperator.EQUAL, 0, 0);

    brg = brgList.get(0);
    op = brg.getOperation("getApprover");
    ruleList = op.getBusinessRulesByName(
         "getApprover", QueryOperator.EQUAL,
         0, 0);

    ruleSet = (RuleSet) ruleList.get(0);
    out.print(RuleArtifactUtility.printRuleSet(rule
    Set));
    }
  }
 } catch (BusinessRuleManagementException e)
 {
  e.printStackTrace();
  out.println(e.getMessage());
 }
 return out.toString();
 }
}
```

Exemple

Sortie du navigateur Web pour l'exemple 11.

```
Exécution de l'exemple 11

Jeu de règles avant publication :
Jeu de règles
Nom : getApprover
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Règle : LargeOrderApprover
Nom affiché : LargeOrderApprover
Description : null
Présentation utilisateur détaillée : Si le nombre d'articles commandés excède 10 et que
la commande excède 5 000 $, l'approbation du superviseur est nécessaire
Présentation utilisateur : Si le nombre d'articles commandés excède {0} et que
la commande excède {1} $, l'approbation du {2} est nécessaire
Nom du paramètre : param0
Valeur du paramètre : 10
Nom du paramètre : param1
Valeur du paramètre : 5000
Nom du paramètre : param2
Valeur du paramètre : superviseur
Règle : DefaultApprover
Nom affiché : DefaultApprover
Description : null
Présentation utilisateur détaillée : approver = peer
Présentation utilisateur : approver = {0}
Nom du paramètre : param0
Valeur du paramètre : peer

Jeu de règles après publication :
Jeu de règles
Nom : getApprover
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
Règle : LargeOrderApprover
Nom affiché : LargeOrderApprover
Description : null
Présentation utilisateur détaillée : Si le nombre d'articles commandés excède 10 et que
la commande excède 5 000 $, l'approbation du superviseur est nécessaire
Présentation utilisateur : Si le nombre d'articles commandés excède {0} et que
```

```
la commande excède {1} $, l'approbation du {2} est nécessaire
Nom du paramètre : param0
Valeur du paramètre : 10
Nom du paramètre : param1
Valeur du paramètre : 5000
Nom du paramètre : param2
Valeur du paramètre : superviseur
Règle : DefaultApprover
Nom affiché : DefaultApprover
Description : null
Présentation utilisateur détaillée : approver = peer
Présentation utilisateur : approver = {0}
Nom du paramètre : param0
Valeur du paramètre : peer
Règle : ExtraLargeOrder
Nom affiché :
Description : null
Présentation utilisateur détaillée : Si le nombre d'articles commandés excède 20 et que
la commande excède 7 500 $, l'approbation du responsable de niveau 2 est nécessaire
Présentation utilisateur : Si le nombre d'articles commandés excède {0} et que
la commande excède {1} $, l'approbation du {2} est nécessaire
Nom du paramètre : param0
Valeur du paramètre : 20
Nom du paramètre : param1
Valeur du paramètre : 7500
Nom du paramètre : param2
Valeur du paramètre : responsable de niveau 2
```

Exemple 12 : Modifier et publier un modèle d'une table de décision en changeant la valeur d'un paramètre

Dans cet exemple, une condition et une action (toutes deux définies avec des modèles) sont modifiées dans une table de décision, en changeant les valeurs des paramètres avant publication.

La méthode la plus simple pour modifier des conditions et des actions dans une table de décision consiste à utiliser des noms uniques pour les modèles à chaque niveau de condition et pour chaque action. Cela permet d'effectuer des recherches sur les noms uniques, puis d'apporter des modifications aux instances de modèle définies à partir de ce modèle. Lorsque des modifications sont apportées à une instance d'un modèle particulier, toutes les valeurs de condition définies avec ce modèle à ce niveau seront mises à jour. Pour les expressions d'action, chaque instance est unique et les modifications apportées à une instance n'affectent pas les autres instances.

Pour cet exemple, un certain nombre de méthodes supplémentaires ont été créées pour simplifier la localisation d'un cas spécifique pour mise à jour, la recherche de la valeur de paramètre spécifique, et la recherche de l'expression d'action définie avec un modèle spécifique.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
```

```
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example12 {
static Formatter out = new Formatter();

static public String executeExample12()
{
 try
 {
   out.clear();
   // Extraction d'un groupe de règles métier par nom et espace de nom
   cible
   List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
    "http://BRSamples/com/ibm/websphere
    /sample/brules",
    QueryOperator.EQUAL,
    "ConfigurationValues",
    QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0)
   {
    // Extraction du premier groupe de règles métier de la liste
    // Ce doit être le seul groupe de règles métier de la
    liste car
    // les combinaisons nom/espace de nom sont
    uniques
    BusinessRuleGroup brg = brgList.get(0);

    // Extraction de l'opération du groupe de règles métier qui
    // la règle métier qui sera modifiée lorsque les
    // règles métier seront associées à une opération
    // spécifique
    Operation op = brg.getOperation("getMessages");

    // Extraction de toutes les règles métier disponibles pour cette
    opération
    List<BusinessRule> ruleList =
    op.getAvailableTargets();

    // Pour cette opération, il n'existe qu'une seule règle métier
    et
    // il s'agit de celle que nous souhaitons mettre à jour
    DecisionTable decisionTable = (DecisionTable)
    ruleList.get(0);
    out.println("");
    out.printlnBold("Table de décision avant publication:");
    out
     .print(RuleArtifactUtility
     .printDecisionTable(decisionT
      able));
```

La règle, les conditions et les actions sont contenues dans une arborescence. Il est possible d'extraire le noeud racine de l'arborescence.

```
    // Extraction de l'arborescence contenant toutes les
    conditions
    // et les actions pour la table de décision
```

```
TreeBlock treeBlock = decisionTable.getTreeBlock();
// Dans l'arborescence, extraction du noeud qui
constitue
// le point de départ pour la navigation dans la table de
décision
TreeNode treeNode = treeBlock.getRootNode();
```

La condition à mettre à jour a été définie à partir d'un modèle appelé “Condition Value Template 2.1”. La méthode `getCaseEdge` permet d'effectuer des recherches récursives à partir du noeud jusqu'au niveau cas, afin de localiser le modèle. Cette méthode suppose que le niveau auquel le modèle est défini soit connu, ainsi que le niveau actuel. Elle peut être utilisée pour rechercher le cas associé à un modèle donné, au cas où un même nom soit utilisé pour différents cas.

```
// Extraction du cas au niveau 1 sous la racine, associé
// à un modèle spécifique avec une valeur de paramètre portant un nom
// spécifique. Etant donné que nous partons d'en haut,
// la profondeur actuelle est 0
CaseEdge caseEdge = getCaseEdge(treeNode, "param0",
"Condition Value Template 2.1", 1, 0);
```

A partir du cas trouvé, il est possible d'extraire l'objet `ConditionValueTemplateInstance` pour la condition.

```
if (caseEdge != null)
{
 // Cas localisé. Extraction de la
 définition de valeur
 // du cas
 TreeConditionValueDefinition condition =
  caseEdge
  .getValueDefinition();
 // Extraction de l'expression de condition définie à l'aide d'un
  correspondant
  TemplateInstanceExpression conditionExpression
  = condition
   .getConditionValueTemplateInstance(
 );
```

Avec l'objet `ConditionValueTemplateInstance`, la valeur de paramètre appropriée peut être extraite, puis mise à jour à l'aide de la méthode `getParameterValue`.

```
 // Extraction du modèle pour l'expression
 Template conditionTemplate =
 conditionExpression
  .getTemplate();

  // Vérification du modèle car il est possible
  d'avoir
  // plusieurs modèles pour une valeur de condition,
  mais un seul peut être
  // appliqué
  if (conditionTemplate.getName().equals(
   "Condition Value Template 2.1"))
  {
   // Extraction de la valeur de paramètre
   ParameterValue parameterValue =
   getParameterValue("param0",
    conditionExpression);

   // Définition de la nouvelle valeur de paramètre
   parameterValue.setValue("info");
  }
```

Il est alors possible d'extraire les différentes expressions d'action définies à l'aide de modèles, afin de les mettre à jour. La méthode `getActionExpressions` renvoie toutes les actions définies avec le modèle Action Value Template 1.

```
ConditionNode conditionNode = (ConditionNode)
treeNode;

// Extraction de l'arborescence de cas
ListCaseEdge> caseEdges =
conditionNode.getCaseEdges();

// Création d'une liste contenant toutes les expressions
d'action qui devront
// également être mises à jour. Etant donné que chaque
action est
// indépendante des autres actions même si elles partagent
le même modèle,
// toutes les actions doivent être mises à jour.
List<TemplateInstanceExpression> expressions =
new Vector<TemplateInstanceExpression>();

// Extraction de toutes les expressions
pour (CaseEdge edge : caseEdges)
{
  getActionExpressions("Action Value
  Template 1", edge,
  expressions);
}
```

Avec la liste des expressions d'action, chaque élément peut être mis à jour. Pour les expressions d'action définies à partir de modèles, la valeur de paramètre appropriée peut être mise à jour.

```
// Mise à jour du paramètre approprié dans chaque
expression
pour (TemplateInstanceExpression expression
expressions)
{
 for (ParameterValue parameterValue :
 expression
  .getParameterValues())
 {
  // Vérification du paramètre
  bien qu'il n'y ait
  // qu'un seul paramètre dans
  notre modèle
  if
   (parameterValue.getParameter().getN
   ame().equals("param0")) {
    String value =
    parameterValue.getValue();
    parameterValue.setValue("Info
    "
    +
    value.substring(value.
    indexOf(":"),
    value.length()));
  }
 }
}
// Une fois la valeur de condition et les actions
mises à jour, le
// groupe de règles métier peut être publié.
// Utilisez la liste d'origine ou créez une liste de
// groupes de règles métier
List<BusinessRuleGroup> publishList = new
ArrayList<BusinessRuleGroup>();
```

```
   // Ajout du groupe de règles métier modifié à la
   liste
   publishList.add(brg);

   // Publication de la liste avec le groupe de règles métier
   mis à jour
   BusinessRuleManager.publish(publishList, true);

   out.println("");

   // Extraction des groupes de règles métier pour
   s'assurer que
   // les modifications ont été publiées
   out.printlnBold("Table de décision après
   publication:");

   brgList =
   BusinessRuleManager.getBRGsByTNSAndName(
    "http://BRSamples/com/ibm/websphere
    /sample/brules",
    QueryOperator.EQUAL,
    "ConfigurationValues",
    QueryOperator.EQUAL, 0, 0);

   brg = brgList.get(0);
   op = brg.getOperation("getMessages");
   ruleList = op.getAvailableTargets();

   decisionTable = (DecisionTable)
   ruleList.get(0);
   out.print(RuleArtifactUtility
    .printDecisionTable(decisionTable))
    ;
    }
   }
 } catch (BusinessRuleManagementException e)
 {
   e.printStackTrace();
   out.println(e.getMessage());
 }
 return out.toString();
}

/*
Méthode permettant de naviguer de façon récursive dans une table de décision
et de localiser un cas associé à un modèle portant un nom spécifique
et contenant un paramètre spécifique à modifier.
Cette méthode suppose que le niveau (depth) auquel se trouve la valeur
à modifier dans la table de décision est connu, et que le niveau actuel
(currentDepth) est connu aussi *
*/
static private CaseEdge getCaseEdge(TreeNode node, String pName,
  String templateName, int depth, int currentDepth)
{
 // Vérification de l'activité du noeud actuel. Ceci indique que
 // cette branche de la table de décision a été entièrement analysée dans le cadre
 // de la recherche de cas
 if (node instanceof ActionNode)
 {
  return null;
 }

 // Extraction des cas pour ce noeud
 List<CaseEdge> caseEdges = ((ConditionNode) node).getCaseEdges();
 for (CaseEdge caseEdge : caseEdges)
 {
```

```
 // Vérification afin de savoir si le niveau approprié a été atteint
 if (currentDepth < depth)
  {
   // Descente d'un niveau et appel de getCaseEdge
   pour
   // traiter ce niveau
   currentDepth++;
   return getCaseEdge(caseEdge.getChildNode(), pName,
    templateName, depth, currentDepth);
  } else
  {
   // Le niveau approprié a été atteint. Extraction
   de la condition pour
   // vérifier si les modèles de cette condition
   correspondent
   // au modèle recherché
   TreeConditionValueDefinition condition = caseEdge
    .getValueDefinition();

   // Extraction de l'expression pour la condition qui a
   été définie
   // avec un modèle
   TemplateInstanceExpression expression = condition
    .getConditionValueTemplateInstance();
   // Extraction du modèle dans l'expression
      Template template = expression.getTemplate();

   // Vérification afin de déterminer si le modèle trouvé est bien celui recherché
    if (template.getName().equals(templateName))
    {
     // Le modèle trouvé est bien celui recherché
      return caseEdge;
    } else
     caseEdge = null;
   }
 }
 return null;
}

/*
Cette méthode permet de rechercher une expression dans les différentes valeurs
de paramètre et si cette expression est trouvée, de renvoyer la valeur
de paramètre concernée.
*/
private static ParameterValue getParameterValue(String pName,
  TemplateInstanceExpression expression)
{
 // Vérification pour s'assurer que l'expression n'est pas nulle,
car une valeur nulle indiquerait
 // que l'expression qui a été transmise n'a probablement pas été
 définie
 // avec un modèle et qu'il n'y a donc aucun paramètre à vérifier.
 if (expression != null) {
  // Extraction des valeurs de paramètre pour l'expression
  List<ParameterValue> parameterValues = expression
    .getParameterValues();

  for (ParameterValue parameterValue : parameterValues)
  {
   // Vérification pour s'assurer que les différents paramètres
   correspondent à la valeur
   // de paramètre recherchée

   if
    (parameterValue.getParameter().getName().equals(pName
   ))
```

```
   {
    // Retour de la valeur de paramètre appropriée
      return parameterValue;
   }
  }
 }
return null;
}
/*
Cette méthode permet de trouver toutes les expressions d'action définies
avec un modèle spécifique. Elle fonctionne de manière récursive
et ajoute les expressions d'action qui correspondent au
paramètre d'expression.
*/

private static void getActionExpressions(String templateName,
CaseEdge next, List<TemplateInstanceExpression>
expressions)
{
 ActionNode actionNode = null;
 TreeNode treeNode = next.getChildNode();

 // Vérification de l'activité du noeud actuel.
 if (treeNode instanceof ConditionNode)
 {
  List<CaseEdge> caseEdges = ((ConditionNode) treeNode)
   .getCaseEdges();

  Iterator<CaseEdge> caseEdgesIterator =
  caseEdges.iterator();

  // Analyse de tous les cas pour trouver les expressions
  // d'action
  while (caseEdgesIterator.hasNext())
   {
    getActionExpressions(templateName,
    caseEdgesIterator.next(),
     expressions);
   }
 } else {
   // ActionNode trouvé
   actionNode = (ActionNode) treeNode;

   List<TreeAction> treeActions = actionNode.getTreeActions();
   // Vérification de la présence d'au moins un élément treeAction
   pour
   // l'expression et analyse des expressions pour vérifier
   // si elles ont été définies avec le modèle spécifique
   // indiqué.
   if (!treeActions.isEmpty())
   {

    Iterator<TreeAction> iterator =
    treeActions.iterator();

   while (iterator.hasNext())
    {
     TreeAction treeAction = iterator.next();
     TemplateInstanceExpression expression =
     treeAction
      .getValueTemplateInstance();

     Template template = expression.getTemplate();

     if (template.getName().equals(templateName))
     {
      // Expression trouvée avec modèle
```

```
      correspondant
      expressions.add(expression);
     }
    }
   }
  }
 }
}
```

Exemple

Sortie du navigateur Web pour l'exemple 12.

```
Exécution de l'exemple 12

Jeu de règles avant publication :
Table de décision
Nom : getMessages
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules

Table de décision après publication :
Table de décision
Nom : getMessages
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
```

Exemple 13 : Ajout d'une valeur de condition et d'actions dans une table de décision

Dans cet exemple, une valeur de condition et une action vont être ajoutées à une table de décision. Pour ajouter une valeur de condition à une table de décision, vous pouvez utiliser un modèle.

Lorsque vous ajoutez une valeur de condition à un noeud de condition, vous ajoutez un cas. Ce nouveau cas est ajouté à la fin de la liste de cas. Pour la valeur de condition, vous devez spécifier une expression d'instance de modèle qui présente les valeurs de paramètre appropriées. Pour spécifier l'expression d'instance de modèle, vous devez utiliser un modèle spécifique. Il est recommandé de choisir des noms uniques pour les modèles à chaque niveau de noeud de condition, afin de pouvoir retrouver les modèles appropriés pour chaque type de condition. Si une définition de modèle unique est utilisée, il peut s'avérer difficile de déterminer le niveau auquel la condition est ajoutée.

Lorsque vous définissez une valeur de condition pour un noeud de condition, vous ajoutez une valeur de condition avec la même instance de modèle pour tous les noeuds de condition de même niveau. Cela est effectué dans le cadre de l'équilibrage de la table de décision. Lorsqu'une valeur de condition est ajoutée, de nouveaux noeuds d'action sont également ajoutés. Ces noeuds d'action comportent trois actions, qui ont des valeurs null pour la présentation utilisateur et pour l'expression d'instance de modèle. Etant donné que la valeur de condition peut être ajoutée à un noeud de condition qui n'a pas de noeud d'action en tant que noeud enfant, l'ajout d'un noeud de condition peut entraîner la création d'un grand nombre de noeuds d'action. Le nombre de noeuds d'action est basé sur le niveau auquel le noeud de condition est ajouté, et sur le nombre de noeuds de condition à ce niveau ainsi que sur le niveau et le nombre de noeuds de condition au niveau enfant.

Pour localiser les noeuds d'action qui ont été créés, vous pouvez effectuer une recherche sur les noeuds d'action avec des actions d'arborescence qui ont des valeurs null pour les présentations utilisateur et les expressions d'instance de

modèle. La méthode TreeActionValueTemplate peut être utilisée pour créer une expression qui peut être définie dans TreeAction. Cette opération doit être répétée pour tous les nouveaux noeuds d'action.

Dans cet exemple, deux méthodes sont fournies pour définir les nouvelles actions d'arborescence. La méthode getEmptyActionNode permet de rechercher de façon récursive un noeud d'action vide à partir du noeud de condition en cours et la méthode getParameterValue permet de renvoyer la valeur d'un paramètre qui a été spécifié par son nom.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionTermDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionValueTemplate;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueTemplate;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;

public class Example13
{
 static Formatter out = new Formatter();

 static public String executeExample13()
 {
  try
  {
   out.clear();

   // Extraction d'un groupe de règles métier par nom et espace de nom
   // cible
   List<BusinessRuleGroup> brgList = BusinessRuleManager
    .getBRGsByTNSAndName(
   "http://BRSamples/com/ibm/websphere/sample/brules",
     QueryOperator.EQUAL,"ConfigurationValues",
     QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0)
    {
     // Extraction du premier groupe de règles métier de la
     // liste. Ce doit être le seul groupe de règles métier
     // de la liste car les combinaisons nom/espace
     // de nom sont uniques
     BusinessRuleGroup brg = brgList.get(0);

     // Extraction de l'opération du groupe de règles métier
```

```
// qui comporte la règle métier qui sera
// modifiée lors de l'association des règles métier
// avec une opération spécifique
Operation op = brg.getOperation("getMessages");

// Extraction de toutes les règles métier disponibles pour
// cette opération
List<BusinessRule> ruleList =
 op.getAvailableTargets();

// Pour cette opération, il n'existe qu'une seule règle
// métier et il s'agit de celle que nous souhaitons
// mettre à jour

DecisionTable decisionTable = (DecisionTable)
 ruleList.get(0);
 out.println1nBold("Table de décision avant
 publication:");
 out.print(RuleArtifactUtility
 .printDecisionTable(decisionTable));
```

Vous devez localiser le niveau auquel la valeur de condition va être ajoutée. Cette information est généralement transmise en tant que paramètre, afin que l'interface utilisateur ou l'application qui utilise les classes sache où ajouter la condition.

```
// Extraction du bloc d'arborescence contenant toutes les
// conditions et les actions pour la table de
// décision
TreeBlock treeBlock =
 decisionTable.getTreeBlock();

// Dans le bloc d'arborescence, extraction du noeud qui
// constitue le point de départ pour la navigation dans
// la table de décision
ConditionNode conditionNode = (ConditionNode)
 treeBlock.getRootNode();

// Extraction des cas pour ce noeud, qui est
// le premier niveau de conditions
List<CaseEdge> caseEdges =
 conditionNode.getCaseEdges();

// Extraction du cas auquel la nouvelle condition
// sera ajoutée
CaseEdge caseEdge = caseEdges.get(0);

// Pour le cas, extraction du noeud de condition afin
// d'extraire les modèles pour la
// condition
conditionNode = (ConditionNode)
    caseEdge.getChildNode();

// Extraction des modèles pour la condition
List<TreeConditionValueTemplate>
 treeValueConditionTemplates = conditionNode
 .getAvailableValueTemplates();

Iterator<TreeConditionValueTemplate>
 treeValueConditionTemplateIterator =
 treeValueConditionTemplates.iterator();

TreeConditionValueTemplate conditionTemplate =
 null;
```

En utilisant des noms de modèle uniques pour chaque niveau de noeud de condition dans la table de décision, vous pouvez vous assurer que la valeur de condition est ajoutée au noeud de condition approprié.

```
// Recherche du modèle à utiliser
 while
(treeValueConditionTemplateIterator.hasNext())
{
 conditionTemplate =
  treeValueConditionTemplateIterator
   .next();
 if (conditionTemplate.getName().equals(
  "Condition Value Template
  2.1"))
  {
   // Modèle trouvé
   break;
  }
  conditionTemplate = null;
}
if (conditionTemplate != null)
{
```

Une fois que vous avez trouvé le modèle approprié, une instance peut être créée et la valeur de paramètre appropriée peut être définie avant l'ajout au noeud de condition.

```
// Extraction de la définition de paramètre à partir
// du modèle
Parameter conditionParameter =
conditionTemplate.getParameter("param0");

// Création d'une instance de valeur de paramètre à
// utiliser dans une nouvelle instance de modèle
// de condition
ParameterValue conditionParameterValue =
  conditionParameter
  .createParameterValue("fatal");

List<ParameterValue>
  conditionParameterValues = new
  ArrayList<ParameterValue>();

// Ajout de la valeur de paramètre à une liste

conditionParameterValues
 .add(conditionParameterValue);

// Création d'une instance de modèle de condition
// avec cette valeur de paramètre
TemplateInstanceExpression
 newConditionValue =
 conditionTemplate
  .createTemplateInstanceExpression(c
 onditionParameterValues);
// Ajout de l'instance de modèle de condition à
// ce noeud de condition
conditionNode

.addConditionValueToThisLevel(newConditionValue);
// Lorsqu'un noeud de condition est ajouté,
// de nouveaux noeuds d'action vides sont
// créés. Il faut ensuite leur ajouter des
// instances de modèle d'action. En exécutant
// une recherche sur les noeuds d'action vides
```

```
// à partir du niveau parent, vous pouvez localiser
// tous les nouveaux noeuds d'action vides.
conditionNode = (ConditionNode)
conditionNode.getParentNode();
```

Une fois la valeur de condition ajoutée au noeud de condition, les actions d'arborescence dans les nouveaux noeuds d'action doivent être définies via la méthode `TreeActionValueTemplate`. Tous d'abord, vous devez localiser le noeud d'action vide pour chaque cas. Utilisez le noeud de condition parent pour vous assurer que, lors des itérations dans les différents noeuds de condition, vous récupérerez tous les noeuds d'action.

```
// Extraction des cas pour le noeud parent
caseEdges = conditionNode.getCaseEdges();

Iterator<CaseEdge> caseEdgesIterator =
caseEdges.iterator();

while (caseEdgesIterator.hasNext())
{
 // Pour chaque cas, extraction d'un
 // noeud d'action vide s'il en existe un
 ActionNode actionNode =
 getEmptyActionNode(caseEdgesIterator
 .next());

 // Vérification pour s'assurer que toutes les actions sont remplies
 if (actionNode != null)
 {
```

Lorsqu'un noeud d'action avec des actions d'arborescence vides est localisé, l'action d'arborescence doit être définie via la méthode `TreeActionValueTemplate`. Tout d'abord, localisez le modèle, puis spécifiez les paramètres avant de créer une instance de modèle. Une fois l'instance de modèle créée, l'action d'arborescence peut être mise à jour. Pour cet exemple, le paramètre a été défini sur une valeur issue d'une autre action d'arborescence d'un autre noeud d'action, sous le même noeud de condition. Pour les autres tables de décision pour lesquelles une autre action d'arborescence n'aura peut-être pas une valeur susceptible d'être utilisée pour créer les nouvelles valeurs de paramètre, cette valeur devra être transmise en tant que paramètre à partir de l'application.

```
 // Extraction de la liste
 // d'actions d'arborescence. Il ne
 // s'agit pas des actions
 // elles-mêmes, mais
 // des marques de
 // réservation
 List<TreeAction>
 treeActionList = actionNode
  .getTreeActions();

 List<TreeActionTermDefinition>
 treeActionTermDefinitions =
  treeBlock
 .getTreeActionTermDefinitions();

 List<TreeActionValueTemplate>
  treeActionValueTemplates =
  treeActionTermDefinitions
  .get(0).getValueTemplates();

 TreeActionValueTemplate
  actionTemplate = null;

 for (TreeActionValueTemplate
```

```
tempActionTemplate :
treeActionValueTemplates)
{

if
(tempActionTemplate.get
 Name().equals(
 "Action Value
 Template 1"))
 {
  actionTemplate =
  tempActionTemplate;
  break;
 }
}

if (actionTemplate != null)
{
 // Extraction d'une autre action
 // qui se trouve sous
 // le noeud de condition
 // parent afin
 // d'utiliser la valeur comme
 // base pour le
 // message d'erreur dans
 // le nouveau
 // noeud d'action. Remontez
 // d'abord jusqu'au
 // noeud de condition
 // parent
 ConditionNode
 parentNode =
 (ConditionNode)
 actionNode
  .getParentNode();

 // Extraction du premier
 // cas du noeud
 // parent, car cette
 // action sera
 // remplie au fur et à mesure
 // de l'ajout de nouvelles
 // actions à la fin
 // de la liste de
 // cas.
 CaseEdge caseE =
  parentNode.getCas
  eEdges().get(
  0);

 // Le noeud enfant est un
 // noeud d'action
 // et se trouve au même
 // niveau que le nouveau
 // noeud d'action.
 ActionNode aNode =
 (ActionNode) caseE
  .getChildNode();

 // Extraction de la liste d'actions
 // d'arborescence
 TreeAction
  existingTreeAction =
  aNode
  .getTreeActions()
  .get(0);
```

```
// Extraction de l'expression
// d'instance
// de modèle pour
// l'action d'arborescence
// à partir de laquelle
// vous pouvez extraire le
// paramètre

TemplateInstanceExpression
existingExpression =
 existingTreeAction
 .getValueTemplateInstance();

ParameterValue
existingParameterValue =
getParameterValue(
  "param0",
existingExpression);

String actionValue =
existingParameterValue
 .getValue();

// Création du nouveau
// message à partir du
// message de
// l'action d'arborescence
// l'action d'arborescence
actionValue = "Fatal"
  +
actionValue.substring(actionValue
 .indexOf(":"), actionValue
 .length());
 Parameter
 actionParameter =
 actionTemplate
 .getParameter("param0");

 // Extraction du paramètre
 // à partir du modèle
 ParameterValue
 actionParameterValue =
  actionParameter
 .createParameterValue(actionValue);

 // Ajout du paramètre à
 // une liste de modèles
 List<ParameterValue>
 actionParameterValues = new
 ArrayList<ParameterValue>();

actionParameterValues.add(actionParameterValue);

 // Création d'une nouvelle
 // instance d'action d'arborescence

 TemplateInstanceExpression
  treeAction = actionTemplate
.createTemplateInstanceExpression(actionParameterValues);

 // Définition de l'action d'arborescence
 // dans le noeud d'action
 // en la définissant dans la
 // liste d'actions d'arborescence
```

Ici, l'action d'arborescence dans le noeud d'action est mise à jour.

```
          treeActionList.get(0)
          .setValueTemplateInstance(
          treeAction);
          }
         }
       }
    }
    // Une fois la valeur de condition et les actions
    // mises à jour, le groupe de règles métier peut être
    // publié.
    // Utilisez la liste d'origine ou créez une nouvelle liste
    // de groupes de règles métier
     List<BusinessRuleGroup> publishList = new
      ArrayList<BusinessRuleGroup>();

    // Ajout du groupe de règles métier modifié à la
    // liste
     publishList.add(brg);

    // Publication de la liste contenant le groupe de règles
    // métier mis à jour

    BusinessRuleManager.publish(publishList, true);

    brgList =
     BusinessRuleManager.getBRGsByTNSAndName(
     "http://BRSamples/com/ibm/websphere/sample/brules",
     QueryOperator.EQUAL, "ConfigurationValues",
     QueryOperator.EQUAL, 0, 0);
     brg = brgList.get(0);
     op = brg.getOperation("getMessages");
     ruleList = op.getAvailableTargets();
     decisionTable = (DecisionTable)
      ruleList.get(0);
     out.println1nBold("Table de décision après
      publication:");
     out
      .print(RuleArtifactUtility
       .printDecisionTable(decisionTable));
    }
  } catch (ValidationException e)
  {
   List<Problem> problems = e.getProblems();

   out.println("Incident = " +
   problems.get(0).getErrorType().name());

   e.printStackTrace();
   out.println(e.getMessage());
  } catch (BusinessRuleManagementException e)
  {
   e.printStackTrace();
   out.println(e.getMessage());
  }
  return out.toString();
}

/*
* Cette méthode permet de rechercher le cas actuel pour tous
* les noeuds d'action qui ont des actions d'arborescence vides. Pour trouver
* un noeud d'action vide, vous devez examiner la fin de la liste
* de cas et vérifier si le noeud d'action comporte des actions d'arborescence
* qui ont des présentations utilisateur et des expressions
* TemplateInstanceExpression nulles.
*/
private static ActionNode getEmptyActionNode(CaseEdge next)
{
```

```
 ActionNode actionNode = null;
 TreeNode treeNode = next.getChildNode();

if (treeNode instanceof ConditionNode)
{
 List<CaseEdge> caseEdges = ((ConditionNode) treeNode)
  .getCaseEdges();

 if (caseEdges.size() > 1)
 {
  // Extraction du cas situé complètement à droite en tant que
  // nouvelle condition. Les actions vides se situent donc complètement à droite
  // des cas
  actionNode = getEmptyActionNode(caseEdges
   .get(caseEdges.size() - 1));

  if (actionNode != null)
  {
   return actionNode;
  }
}
} else
{
 actionNode = (ActionNode) treeNode;

 List<TreeAction> treeActions =
 actionNode.getTreeActions();

 if (!treeActions.isEmpty())
 {
 if
 ((treeActions.get(0).getValueUserPresentation() == null)
  &&
 (treeActions.get(0).getValueTemplateInstance() == null))
  {
   return actionNode;
  }
 }
  `actionNode = null;
 }
 return actionNode;
}
/*
 * Cette méthode permet de rechercher une expression dans les différentes
valeurs de paramètre
 * et si cette expression est trouvée, de renvoyer la valeur de paramètre
 * concernée.
 */
 private static ParameterValue getParameterValue(String pName,
   TemplateInstanceExpression expression)
 {
  ParameterValue parameterValue = null;

  // Vérification pour s'assurer que l'expression n'est pas nulle,
car une valeur nulle indiquerait
  // que l'expression qui a été transmise n'a probablement pas été
  // définie avec un modèle et qu'il n'y a donc aucun
  // paramètre à vérifier.
  if (expression != null)
  {
   // Extraction des valeurs de paramètre pour l'expression
   List<ParameterValue> parameterValues = expression
    .getParameterValues();
   Iterator<ParameterValue> parameterIterator =
    parameterValues
    .iterator();
```

```
   // Vérification pour s'assurer que les différents paramètres
   // correspondent à la valeur de paramètre recherchée
   while (parameterIterator.hasNext())
   {
    parameterValue = parameterIterator.next();

    if
    (parameterValue.getParameter().getName().equals(pName))
    {
    // Retour de la valeur de paramètre
    // appropriée
    return parameterValue;
    }
  }
 }
 return parameterValue;
}
}
```

Exemple

Sortie du navigateur Web pour l'exemple 13.

```
Exécution de l'exemple 13

Table de décision avant publication :
Table de décision
Nom : getMessages
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules

Table de décision après publication :
Table de décision
Nom : getMessages
Espace de nom : http://BRSamples/com/ibm/websphere/sample/brules
```

Exemple 14 : Gestion des erreurs dans un jeu de règles

Cet exemple explique comment identifier des incidents dans un jeu de règles et déterminer la nature de l'incident, afin d'afficher le message approprié ou de mettre en oeuvre l'action nécessaire pour corriger la situation.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import
com.ibm.wbiserver.brules.mgmt.problem.ProblemStartDateAfterEndDate;
import com.ibm.wbiserver.brules.mgmt.problem.ValidationError;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example14 {
static Formatter out = new Formatter();
```

```
static public String executeExample14() {
 try {
   out.clear();

   // Extraction d'un groupe de règles métier par nom et espace de nom
   cible
   List<BusinessRuleGroup> brgList = BusinessRuleManager
     .getBRGsByTNSAndName(
     "http://BRSamples/com/ibm/websphere
     /sample/brules",
     QueryOperator.EQUAL,
     "ApprovalValues",
     QueryOperator.EQUAL, 0, 0);

   if (brgList.size() > 0) {
     // Extraction du premier groupe de règles métier de la liste
     // Cela doit être le seul groupe de règles métier de la
     liste car
     // les combinaisons nom/espace de nom sont
     uniques
     BusinessRuleGroup brg = brgList.get(0);
     out.println("Groupe de règles métier extrait");

     // Extraction de l'opération du groupe de règles métier qui
     // comporte la règle métier qui sera modifiée lorsque les
     // règles métier seront associées à une opération
     // spécifique
     Operation op = brg.getOperation("getApprover");

     // Extraction d'une règle spécifique par son nom
     List<BusinessRule> ruleList =
     op.getBusinessRulesByName(
       "getApprover", QueryOperator.EQUAL, 0,
       0);

     // Extraction de la règle spécifique
     RuleSet ruleSet = (RuleSet) ruleList.get(0);
     out.println("Jeu de règles extrait");

     RuleBlock ruleBlock = ruleSet.getFirstRuleBlock();

     Iterator<RuleSetRule> ruleIterator =
     ruleBlock.iterator();

     // Recherche de la règle à
     modifier
     while (ruleIterator.hasNext()) {
       RuleSetRule rule = ruleIterator.next();

       // Vérification pour s'assurer que la règle a été définie avec un
       modèle
       // afin de permettre les modifications.
       if (rule instanceof
       RuleSetTemplateInstanceRule) {
        // Extraction de l'instance de règle du modèle
        RuleSetTemplateInstanceRule
        templateInstance =
        (RuleSetTemplateInstanceRule) rule;
        // Vérification pour s'assurer qu'il s'agit de l'instance de règle de modèle
        appropriée
        if (templateInstance.getName().equals(
          "LargeOrderApprover")) {
```

Pour provoquer un incident, cet exemple définir pour un paramètre une valeur qui n'est pas valide pour l'expression. En effet, le paramètre attend un entier, mais une chaîne est spécifiée.

```
        // Extraction du paramètre de l'instance de
        modèle
        ParameterValue parameter =
        templateInstance
          .getParameterValue("par
          am1");

        // Définition d'une valeur incorrecte pour ce
        paramètre
        // Cela provoque une erreur de
        validation
        parameter.setValue("3500 $");
        out.println("Valeur incorrecte saisie
        pour un paramètre");
        break;
      }
    }
  }
  // Il n'est pas possible d'accéder à ce code en raison
  de l'erreur
  // introduite
  // ci-dessus

  // Une fois la valeur de condition et les actions mises à jour, le
  groupe de
  // règles
  // métier peut être publié.
  // Utilisez la liste d'origine ou créez une nouvelle liste
  // de groupes de règles métier
  List<BusinessRuleGroup> publishList = new
  ArrayList<BusinessRuleGroup>();

  // Ajout du groupe de règles métier modifié à la liste
  publishList.add(brg);

  // Publication de la liste avec le groupe de règles métier
  mis à jour
  BusinessRuleManager.publish(publishList, true);
}
```

Une erreur `ValidationException` est émise et à partir de cette exception, les incidents peuvent être extraits. Pour chaque incident, il est alors possible de déterminer la nature de l'erreur. Un message peut être imprimé ou une action approprié peut être exécutée.

```
} catch (ValidationException e) {
out.println("Erreur de validation");

List<Problem> problems = e.getProblems();

Iterator<Problem> problemIterator = problems.iterator();

// Recherche de l'erreur concernée dans la liste des incidents et
// exécution de l'action appropriée. Par exemple, signaler l'erreur
// ou corriger
while (problemIterator.hasNext()) {
  Problem problem = problemIterator.next();
  ValidationError error = problem.getErrorType();

  // Identification de la valeur de l'erreur
  if (error == ValidationError.TYPE_CONVERSION_ERROR) {
    // Gestion de l'erreur en signalant
    l'incident
      .println("Incident : Valeur incorrecte
      saisie pour un paramètre");
    return out.toString();
  }
```

```
    // else if....
    // Possibilité de rechercher d'autres erreurs et d'imprimer
    // le message correspondant ou d'exécuter l'action
    appropriée pour
    // corriger la situation
  }
} catch (BusinessRuleManagementException e) {
  out.println("Erreur");
  e.printStackTrace();
}
return out.toString();
}
}
```

Exemple

Sortie du navigateur Web pour l'exemple 14.

```
Exécution de l'exemple 14

Groupe de règles métier extrait
Jeu de règles extrait
Erreur de validation
Incident : Valeur incorrecte saisie pour un paramètre
```

Exemple 15 : Gestion des erreurs dans un groupe de règles métier

Cet exemple est similaire à l'exemple 14, car il montre comment gérer les incidents qui peuvent se produire lors de la publication d'un groupe de règles métier. Il montre comment déterminer la nature de l'incident afin d'imprimer le message correspondant ou d'exécuter l'action appropriée.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleGroup;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManagementException;
import com.ibm.wbiserver.brules.mgmt.BusinessRuleManager;
import com.ibm.wbiserver.brules.mgmt.Operation;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecord;
import com.ibm.wbiserver.brules.mgmt.OperationSelectionRecordList;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.ValidationException;
import com.ibm.wbiserver.brules.mgmt.problem.Problem;
import
com.ibm.wbiserver.brules.mgmt.problem.ProblemStartDateAfterEndDate;
import com.ibm.wbiserver.brules.mgmt.query.QueryOperator;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class Example15
{
static Formatter out = new Formatter();

static public String executeExample15()
{
 try
 {
```

```
 out.clear();

// Extraction d'un groupe de règles métier par nom et espace de nom
cible
List<BusinessRuleGroup> brgList = BusinessRuleManager
 .getBRGsByTNSAndName(
 "http://BRSamples/com/ibm/websphere
 /sample/brules",
 QueryOperator.EQUAL,
 "ApprovalValues",
 QueryOperator.EQUAL, 0, 0);
 if (brgList.size() > 0)
 {
  // Extraction du premier groupe de règles métier de la liste
  // Cela doit être le seul groupe de règles métier de la
  liste car
  // la combinaison de nom et d'espace de nom cible est
  unique
  BusinessRuleGroup brg = brgList.get(0);
  out.println("Groupe de règles métier extrait");

  // Extraction de l'opération du groupe de règles métier qui comporte
  // la règle métier qui sera modifiée lorsque les
  // règles métier seront associées à une opération
  // spécifique
  Operation op = brg.getOperation("getApprover");

  // Extraction d'une règle spécifique par son nom
  List<BusinessRule> ruleList =
  op.getBusinessRulesByName(
   "getApprover", QueryOperator.EQUAL, 0,
   0);

  // Extraction de la règle spécifique
  RuleSet ruleSet = (RuleSet) ruleList.get(0);
   out.println("Jeu de règles extrait");

  RuleBlock ruleBlock = ruleSet.getFirstRuleBlock();

  Iterator<RuleSetRule> ruleIterator =
   ruleBlock.iterator();

  // Recherche de la règle à
  modifier
  while (ruleIterator.hasNext())
  {
   RuleSetRule rule = ruleIterator.next();

   // Vérification pour s'assurer que la règle a été définie avec un
   modèle
   // afin de permettre les modifications.
   if (rule instanceof
   RuleSetTemplateInstanceRule)
   {
    // Extraction de l'instance de règle du modèle
    RuleSetTemplateInstanceRule
    templateInstance =
    (RuleSetTemplateInstanceRule) rule;

    // Vérification pour s'assurer qu'il s'agit de l'instance de règle de modèle
    appropriée
    if (templateInstance.getName().equals(
      "LargeOrderApprover"))
    {
     // Extraction du paramètre de l'instance de
     modèle
     ParameterValue parameter =
```

```
      templateInstance
       .getParameterValue("par
       am1");

      // Définition de la valeur de ce paramètre
      // Cette valeur est au format
      approprié et ne
      // provoquera pas d'erreur de validation
      parameter.setValue("4000");
      out.println("Valeur de paramètre de jeu de règle
      définie correctement");
      break;
      }
    }
  }
```

Pour vérifier si un jeu de règles est correct, vous pouvez appeler la méthode validate. La méthode validate est disponible pour tous les objets et renvoie une liste d'incidents permettant d'identifier les erreurs. Lorsque vous appelez la méthode validate pour un objet, elle est également exécutée pour tous les sous-objets qu'il contient.

```
      // Vérification des modifications apportées au jeu de règles
      List<Problem> problems = ruleSet.validate();
      out.println("Jeu de règles validé");

      // Normalement, ce jeu d'essai ne contient aucune erreur,
      // mais recherchez quand même les éventuels problèmes, puis
      // prenez les mesures nécessaires pour corriger ou signaler
      // l'erreur
      if (problems != null)
      {
       Iterator<Problem> problemIterator =
       problems.iterator();

       while (problemIterator.hasNext())
       {
        Problem problem = problemIterator.next();

        if (problem instanceof
        ProblemStartDateAfterEndDate)
        {
         out
         .println("Valeur
          incorrecte saisie pour un
           paramètre");
         return out.toString();
        }
       }
      } else
      {
       out.println("Aucun incident détecté pour le jeu de
       règles");
      }
      // Extraction de la liste des règles cible disponibles
      List<BusinessRule> ruleList2 =
      op.getAvailableTargets();

      // Extraction de la première règle planifiée
      l'élément comportant une
      BusinessRule rule = ruleList2.get(0);

      // Pour créer une condition d'erreur, nous allons définir l'heure de fin
      d'une règle
      // planifiée 1 heure avant l'heure de début
      // Cela provoque une erreur de validation
      Date future = new Date();
```

```
    long futureTime = future.getTime() - 360000;

    // Extraction de la liste de sélection d'opération pour ajouter
    l'élément comportant une
    // erreur de planification
    OperationSelectionRecordList opList = op
    .getOperationSelectionRecordList();

    // Création d'une nouvelle instance de règle planifiée
    // Aucune erreur n'est renvoyée jusqu'à la validation ou la publication,
    // car d'autres modifications peuvent être apportées
    OperationSelectionRecord newRecord = opList
     .newOperationSelectionRecord(new Date(),
     new Date(
     futureTime), rule);
```

Lorsque l'enregistrement est ajouté avec des dates incorrectes, aucune erreur n'est renvoyée. Des chevauchements peuvent se produire ou aucun enregistrement de sélection n'est défini pour l'opération, tandis que des modifications sont en cours. L'erreur sera identifiée lors de la publication du groupe de règles métier comportant l'enregistrement de sélection d'opération en question. La méthode validate est appelée avant la publication des objets et des exceptions sont émises si des erreurs sont identifiées.

```
    // Ajout de l'instance de règle planifiée à l'opération
    // Aucune erreur identifiée
    opList.addOperationSelectionRecord(newRecord);
    out.println("Nouvel enregistrement de sélection ajouté avec
    une planification incorrecte");

    // Une fois la valeur de condition et les actions mises à jour, le
    groupe
    // de règles
    // métier peut être publié.
    // Utilisez la liste d'origine ou créez une nouvelle liste
    // de groupes de règles métier
    List<BusinessRuleGroup> publishList = new
    ArrayList<BusinessRuleGroup>();

    // Ajout du groupe de règles métier modifié à la liste
    publishList.add(brg);

    // Publication de la liste avec le groupe de règles métier
    mis à jour
    BusinessRuleManager.publish(publishList, true);
   }
} catch (ValidationException e) {
out.println("Erreur de validation");

List<Problem> problems = e.getProblems();

Iterator<Problem> problemIterator = problems.iterator();
// Il peut y avoir plusieurs incidents
// Passez en revue tous les incidents, et traitez chacun d'entre eux ou
// signalez l'incident
while (problemIterator.hasNext())
{
 Problem problem = problemIterator.next();

 // Chaque incident est de type différent et il est possible de les
 comparer
 if (problem instanceof ProblemStartDateAfterEndDate)
 {
  out
  .println("La planification de la règle est
   incorrecte. La date de début est postérieure à la date de
```

```
      fin.");
     return out.toString();
     }
    // else if....
    // Possibilité de rechercher d'autres erreurs et d'imprimer
    // le message correspondant ou d'exécuter l'action
    appropriée pour
    // corriger la situation
  }
 }catch (BusinessRuleManagementException e)
 {
  out.println("Erreur");
  e.printStackTrace();
 }
 return out.toString();
 }
}
```

Exemple

Sortie du navigateur Web pour l'exemple 15.

```
Exécution de l'exemple 15

Groupe de règles métier extrait
Jeu de règles extrait
Valeur de paramètre de jeu de règle définie correctement
Jeu de règles validé
Erreur de validation
Planification incorrecte de la règle. La date de début est postérieure à la date de fin.
```

Autres exemples de requêtes

Les exemples suivants ne figurent pas dans l'application contenant les exemples 1 à 15 ; toutefois, ils illustrent la création de requêtes qui permettent d'extraire des groupes de règles métier.

Dans ces exemples, différentes propriétés et caractères génériques (‘_’, ‘%’) sont utilisés avec différents opérateurs (AND, OR, LIKE, NOT_LIKE, EQUAL et NOT_EQUAL).

Exemple

Pour les besoins de ces exemples, les requêtes renverront différentes combinaisons de 4 groupes de règles métier. Il est important de bien comprendre les différents attributs et propriétés des groupes de règles métier, car ils sont utilisés dans les requêtes.

```
Nom : BRG1
Espace de nom cible : http://BRG1/com/ibm/br/rulegroup
Propriétés :
organization, 8JAA
department, claims
ID, 00000567
region: SouthCentralRegion
manager: Joe Bean

Nom : BRG2
Espace de nom cible : http://BRG2/com/ibm/br/rulegroup
Propriétés :
organization, 7GAA
department, accounting
ID, 0000047
ID_cert45, ABC
region: NorthRegion
```

```
Nom : BRG3
Espace de nom cible : http://BRG3/com/ibm/br/rulegroup
Propriétés :
organization, 7FAB
department, finance
ID, 0000053
ID_app45, DEF
region: NorthCentralRegion

Nom : BRG4
Espace de nom cible : http://BRG4/com/ibm/br/rulegroup
Propriétés :
organization, 7HAA
department, shipping
ID, 0000023
ID_app45, GHI
region: SouthRegion
```

Référence associée

Interrogation par une propriété unique
Ceci est un exemple d'interrogation par une propriété unique.

Interrogation de groupes de règles métier par des propriétés et des caractères génériques (%) au début et à la fin de la valeur
Ceci est un exemple d'interrogation de groupes de règles métier par des propriétés et des caractères génériques (%) au début et à la fin de la valeur

Interrogation de groupes de règles métier par des propriétés et un caractère générique ('_')
Ceci est un exemple d'interrogation de groupes de règles métier par des propriétés et un caractère générique ('_').

Interrogation de groupes règles métier par des propriétés avec plusieurs caractères génériques ('_' et '%')
Ceci est un exemple d'interrogation de groupes règles métier par des propriétés avec plusieurs caractères génériques ('_' et '%')

Interrogation de groupes de règles métier par l'opérateur NOT_LIKE et un caractère générique ('_')
Ceci est un exemple d'interrogation de groupes de règles métier par l'opérateur NOT_LIKE et un caractère générique ('_').

Interrogation de groupes de règles métier par l'opérateur NOT_EQUAL
Ceci est un exemple d'interrogation de groupes de règles métier par l'opérateur NOT_EQUAL.

Interrogation de groupes de règles métier par PropertyIsDefined
Ceci est un exemple d'interrogation de groupes de règles métier par PropertyIsDefined.

Interrogation de groupes de règles métier par NOT PropertyIsDefined
Ceci est un exemple d'interrogation de groupes de règles métier par NOT PropertyIsDefined.

Interrogation de groupes de règles métier par plusieurs propriétés avec un noeud NOT unique
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec un noeud NOT unique.

Interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur AND
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur AND.

Interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur OR
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur OR.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec plusieurs opérateurs AND
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec plusieurs opérateurs AND.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et OR
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et OR.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et NOT
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et NOT.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs NOT et OR
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs NOT et OR.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués et un noeud NOT
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués et un noeud NOT.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT.

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT
Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT.

Interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur AND
Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur AND.

Interrogation de groupes de règles métier par une liste de noeuds et un noeud combiné avec un opérateur AND
Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur AND.

Interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur OR
Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur OR.

Interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur OR
Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur OR.

Interrogation par une propriété unique :

Ceci est un exemple d'interrogation par une propriété unique.

```
List<BusinessRuleGroup> brgList = null;

brgList = BusinessRuleManager.getBRGsBySingleProperty(
  "department", QueryOperator.EQUAL,
  "accounting", 0, 0);
// Returns BRG2
```

Interrogation de groupes de règles métier par des propriétés et des caractères génériques (%) au début et à la fin de la valeur :

Ceci est un exemple d'interrogation de groupes de règles métier par des propriétés et des caractères génériques (%) au début et à la fin de la valeur

```
// Query Prop AND Prop
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "region", QueryOperator.LIKE,
  "%Region");

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode(
  "ID", QueryOperator.LIKE,
  "000005%");

QueryNode queryNode =
QueryNodeFactory.createAndNode(leftNode,
  rightNode);

brgList =
BusinessRuleManager.getBRGsByProperties(queryNode, 0, 0);
// Returns BRG1 and BRG3
```

Interrogation de groupes de règles métier par des propriétés et un caractère générique ('_') :

Ceci est un exemple d'interrogation de groupes de règles métier par des propriétés et un caractère générique ('_').

```
brgList = BusinessRuleManager.getBRGsBySingleProperty("ID",
QueryOperator.LIKE, "00000_3", 0, 0);

// Returns BRG3 and BRG4
```

Interrogation de groupes règles métier par des propriétés avec plusieurs caractères génériques ('_' et '%') :

Ceci est un exemple d'interrogation de groupes règles métier par des propriétés avec plusieurs caractères génériques ('_' et '%')

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("region",
QueryOperator.LIKE, "__uth%Region",
  0, 0);

// Returns BRG1 and BRG4
```

Interrogation de groupes de règles métier par l'opérateur NOT_LIKE et un caractère générique ('_') :

Ceci est un exemple d'interrogation de groupes de règles métier par l'opérateur NOT_LIKE et un caractère générique ('_').

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("organization",
QueryOperator.NOT_LIKE,
  "7__A", 0, 0);

// Returns BRG1 and BRG3
brgList =
BusinessRuleManager.getBRGsBySingleProperty("organization",
QueryOperator.NOT_LIKE,
  "7%", 0, 0);

// Returns BRG1
```

Interrogation de groupes de règles métier par l'opérateur NOT_EQUAL :

Ceci est un exemple d'interrogation de groupes de règles métier par l'opérateur NOT_EQUAL.

```
brgList =
BusinessRuleManager.getBRGsBySingleProperty("department",
QueryOperator.NOT_EQUAL,
  "claims", 0, 0);
// Returns BRG1
```

Interrogation de groupes de règles métier par PropertyIsDefined :

Ceci est un exemple d'interrogation de groupes de règles métier par PropertyIsDefined.

```
PropertyIsDefinedQueryNode node =
QueryNodeFactory.createPropertyIsDefinedQueryNode("manager"
);

brgList = BusinessRuleManager.getBRGsByProperties(node, 0,
0);

// Returns BRG1
```

Interrogation de groupes de règles métier par NOT PropertyIsDefined :

Ceci est un exemple d'interrogation de groupes de règles métier par NOT PropertyIsDefined.

```
// NOT Prop
QueryNode node =
QueryNodeFactory.createPropertyIsDefinedQueryNode("manager"
);

NotNode notNode = QueryNodeFactory.createNotNode(node);

brgList = BusinessRuleManager.getBRGsByProperties(notNode,
0, 0);

// Returns BRG1
```

Interrogation de groupes de règles métier par plusieurs propriétés avec un noeud NOT unique :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec un noeud NOT unique.

```
// Prop AND NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.EQUAL, "accounting");
```

```
NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID",
  QueryOperator.LIKE, "00000%");

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
  notNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

Interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur AND :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur AND.

```
// NOT Prop AND NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.EQUAL, "accounting");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "cla%");

NotNode notNode2 =
QueryNodeFactory.createNotNode(leftNode);

AndNode andNode = QueryNodeFactory.createAndNode(notNode,
notNode2);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG1 and BRG2
```

Interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur OR :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés avec plusieurs noeuds NOT combinés avec un opérateur OR.

```
// NOT Prop OR NOT Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department", QueryOperator.EQUAL,
  "claims");

NotNode notNode2 =
QueryNodeFactory.createNotNode(leftNode);
```

```
OrNode orNode = QueryNodeFactory.createOrNode(notNode,
notNode2);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG1, BRG2, BRG3, and BRG4
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec plusieurs opérateurs AND :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec plusieurs opérateurs AND.

```
// (Prop AND Prop) AND (Prop AND Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "7GAA");

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(leftNode,rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("ID",
QueryOperator.LIKE,"000004_");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
QueryOperator.EQUAL,
  "NorthRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft,andNodeRight);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et OR :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et OR.

```
// (Prop AND Prop) OR (Prop AND NOT Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE, "acc%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "7GAA");

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(leftNode,rightNode);

QueryNode rightNode2 =
```

```
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.EQUAL, "8JAA");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE, "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, notNode);

OrNode orNode = QueryNodeFactory.createOrNode(andNodeLeft,
andNodeRight);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG2 and BRG3
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et NOT :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND et NOT.

```
// Prop AND NOT (Prop AND Prop)
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID",
QueryOperator.LIKE, "000005%");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
QueryOperator.EQUAL,
  "8JAA");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
  "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2, rightNode2);

NotNode notNode =
QueryNodeFactory.createNotNode(andNodeRight);

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
notNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG3
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs NOT et OR :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs NOT et OR.

```
// NOT (Prop AND Prop) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
```

```
  "8_A_");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
  "%lRegion");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

NotNode notNode =
QueryNodeFactory.createNotNode(andNodeRight);

OrNode orNode = QueryNodeFactory.createOrNode(notNode,
rightNode);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG3
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

```
// Prop AND (Prop AND (Prop AND Prop))
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

PropertyIsDefinedQueryNode node2 =
QueryNodeFactory.createPropertyIsDefinedQueryNode("ID_cert4
5");

AndNode andNode = QueryNodeFactory.createAndNode(node2,
andNodeLeft);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);
// Returns BRG2
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

```
// (Prop AND (Prop AND Prop)) AND Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",QueryOper
ator.LIKE,
"___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
QueryOperator.LIKE,
"7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
QueryOperator.LIKE,
"%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_app45",QueryOp
erator.LIKE, "GH_");

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft, leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG4
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués et un noeud NOT :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués et un noeud NOT.

```
// Prop AND (Prop AND (Prop AND NOT Prop))
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%lRegion");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
```

```
QueryNodeFactory.createAndNode(leftNode2,notNode);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "AB_");

AndNode andNode = QueryNodeFactory.createAndNode(leftNode,
andNodeLeft);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG2
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs AND imbriqués.

```
// (Prop AND (Prop AND Prop)) AND Prop - Return empty
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

AndNode andNodeRight =
QueryNodeFactory.createAndNode(leftNode2,rightNode2);

AndNode andNodeLeft =
QueryNodeFactory.createAndNode(rightNode,andNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

AndNode andNode =
QueryNodeFactory.createAndNode(andNodeLeft, leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

//Returns no BRGs
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués.

```
// (Prop OR (Prop OR Prop)) OR Prop

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(leftNode2,rightNode2);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(rightNode,orNodeRight);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

OrNode orNode = QueryNodeFactory.createOrNode(orNodeLeft,
leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG1
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués.

```
// (Prop OR (Prop OR NOT Prop)) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(leftNode2,notNode);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(rightNode,orNodeRight);
```

```
QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("ID_cert45",
  QueryOperator.LIKE,
  "GH_");

OrNode orNode = QueryNodeFactory.createOrNode(orNodeLeft,
leftNode);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

// Returns BRG3
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT.

```
// Prop OR NOT(Prop OR Prop)
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "___thRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode(
  "organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(rightNode2,
  rightNode);

NotNode notNode =
QueryNodeFactory.createNotNode(orNodeRight);

OrNode orNodeLeft = QueryNodeFactory.createOrNode(leftNode,
notNode);

brgList =
BusinessRuleManager.getBRGsByProperties(orNodeLeft, 0, 0);

// Returns BRG3
```

Interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT :

Ceci est un exemple d'interrogation de groupes de règles métier par plusieurs propriétés combinées avec des opérateurs OR imbriqués et un noeud NOT.

```
// NOT(Prop OR Prop) OR Prop
QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%lRegion");

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode(
```

```
  "organization",
  QueryOperator.LIKE,
  "7%");

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode(
  "department",
  QueryOperator.LIKE,
  "%ing");

OrNode orNodeRight =
QueryNodeFactory.createOrNode(rightNode2,rightNode);

NotNode notNode =
QueryNodeFactory.createNotNode(orNodeRight);

OrNode orNodeLeft =
QueryNodeFactory.createOrNode(notNode,leftNode);

brgList =
BusinessRuleManager.getBRGsByProperties(orNodeLeft, 0, 0);

// Returns BRG2 and BRG4
```

Interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur AND :

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur AND.

```
// AND list
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7%");

list.add(rightNode2);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "7H%");

list.add(leftNode2);

AndNode andNode = QueryNodeFactory.createAndNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Returns BRG4
```

Interrogation de groupes de règles métier par une liste de noeuds et un noeud combiné avec un opérateur AND :

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur AND.

```
// AND list with a notNode
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

NotNode notNode =

QueryNodeFactory.createNotNode(rightNode2);

list.add(notNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",

list.add(leftNode2);

AndNode andNode = QueryNodeFactory.createAndNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(andNode,
0, 0);

// Return BRG4
```

Interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur OR :

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds combinés avec un opérateur OR.

```
// OR list
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");
```

```
list.add(rightNode2);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

OrNode orNode = QueryNodeFactory.createOrNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG3
```

Interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur OR :

Ceci est un exemple d'interrogation de groupes de règles métier par une liste de noeuds et un noeud NOT combiné avec un opérateur OR.

```
// OR list with Not node
List<QueryNode> list = new ArrayList<QueryNode>();

QueryNode rightNode =
QueryNodeFactory.createPropertyQueryNode("region",
  QueryOperator.LIKE,
  "%thRegion");

list.add(rightNode);

QueryNode rightNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

NotNode notNode =
QueryNodeFactory.createNotNode(rightNode2);

list.add(notNode);

QueryNode leftNode =
QueryNodeFactory.createPropertyQueryNode("department",
  QueryOperator.LIKE,
  "%ing");

list.add(leftNode);

QueryNode leftNode2 =
QueryNodeFactory.createPropertyQueryNode("organization",
  QueryOperator.LIKE,
  "8%");

list.add(leftNode2);

OrNode orNode = QueryNodeFactory.createOrNode(list);

brgList = BusinessRuleManager.getBRGsByProperties(orNode,
0, 0);

//Returns BRG1, BRG2, BRG3, and BRG4
```

Classes d'opérations communes

Cette section contient des classes supplémentaires, qui ont été utilisées dans les exemples pour simplifier des opérations communes.

Référence associée

Classe Formatter
Cette classe fournit diverses méthodes permettant d'afficher les différents exemples. Elle ajoute diverses balises HTML pour formater la sortie.

Classe RuleArtifactUtility
Cette classe utilitaire comporte deux méthodes publiques. La première sert à imprimer une table de décision. Cette méthode exploite une méthode privée qui utilise la récursivité pour imprimer les conditions et les actions de la table de décision. La seconde méthode publique sert à imprimer un jeu de règles.

Classe Formatter

Cette classe fournit diverses méthodes permettant d'afficher les différents exemples. Elle ajoute diverses balises HTML pour formater la sortie.

```
package com.ibm.websphere.sample.brules.mgmt;

public class Formatter {

private StringBuffer buffer;

public Formatter()
{
  buffer = new StringBuffer();
}

public void println(Object o)
{
  buffer.append(o);
  buffer.append("<br>\n");
  }

  public void print(Object o)
  {
    buffer.append(o);
  }

  public void printlnBold(Object o)
  {
    buffer.append("<b>");
    buffer.append(o);
    buffer.append("</b><brbr>\n");
  }

  public void printBold(Object o)
  {
    buffer.append("<b>");
    buffer.append(o);
    buffer.append("</b>");
  }

  public String toString()
  {
    return buffer.toString();
  }

  public void clear()
  {
    buffer = new StringBuffer();
  }
}
```

Classe RuleArtifactUtility

Cette classe utilitaire comporte deux méthodes publiques. La première sert à imprimer une table de décision. Cette méthode exploite une méthode privée qui utilise la récursivité pour imprimer les conditions et les actions de la table de décision. La seconde méthode publique sert à imprimer un jeu de règles.

```
package com.ibm.websphere.sample.brules.mgmt;

import java.util.Iterator;
import java.util.List;

import com.ibm.wbiserver.brules.mgmt.BusinessRule;
import com.ibm.wbiserver.brules.mgmt.Parameter;
import com.ibm.wbiserver.brules.mgmt.ParameterValue;
import com.ibm.wbiserver.brules.mgmt.RuleTemplate;
import com.ibm.wbiserver.brules.mgmt.Template;
import com.ibm.wbiserver.brules.mgmt.dtable.ActionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.CaseEdge;
import com.ibm.wbiserver.brules.mgmt.dtable.ConditionNode;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTable;
import com.ibm.wbiserver.brules.mgmt.dtable.DecisionTableRule;
import
com.ibm.wbiserver.brules.mgmt.dtable.DecisionTableTemplateInstanceRule;
import com.ibm.wbiserver.brules.mgmt.dtable.TemplateInstanceExpression;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeAction;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeActionTermDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeBlock;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionTermDefinition;
import
com.ibm.wbiserver.brules.mgmt.dtable.TreeConditionValueDefinition;
import com.ibm.wbiserver.brules.mgmt.dtable.TreeNode;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleBlock;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSet;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRule;
import com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetRuleTemplate;
import
com.ibm.wbiserver.brules.mgmt.ruleset.RuleSetTemplateInstanceRule;

public class RuleArtifactUtility
{
  static Formatter out = new Formatter();

  /*
  Method to print out a decision table with the conditions and
  actions printed out in a HTML tabular format. The conditions
  and actions are printed out with a separate method that
  recursively works through the case edges of the decision
  tables.
  */

  public static String printDecisionTable(BusinessRule
  ruleArtifact)
  {
    out.clear();
    out.printlnBold("Decision Table");
    DecisionTable decisionTable = (DecisionTable)
    ruleArtifact;
    out.println("Name: " +
    decisionTable.getName());
    out.println("Namespace: " +
    decisionTable.getTargetNameSpace());

    // Output the init rule for the decision table
    before
    // working through the table of conditions and
    actions
```

```
DecisionTableRule initRule =
decisionTable.getInitRule();
if (initRule != null)
{
  out.printBold("Init Rule: ");
  out.println(initRule.getName());
  out.println("Display Name: " +
  initRule.getDisplayName());
  out.println("Description: " +
  initRule.getDescription());
  // The expanded user presentation
  will automatically populate the
  // presentation with the parameter
  values and can be used for
  // display if the init rule was
  defined with a template. If no
  // template was defined the
  expanded user presentation
  // is the same as the regular
  presentation.
  out.println("Extended User
  Presentation: "
    +
  initRule.getExpandedUse
  rPresentation());
  // The regular user presentation
  will have placeholders in the
  // string where the
  // parameter can be substituted if
  the init rule was defined with a
  // template
  // If the rule was not defined with
  a template, the user
  // presentation will only
  // be a string without
  placeholders. The placeholders are
  of a
  // format of {n} where
  // n is the index (zero-based) of
  the parameter in the template. This
  // value
  // can be used to create an
  interface for editing where there
  are
  // fields with
  // the parameter values available
  for editing
  out.println("User Presentation: " +
  initRule.getUserPresentation());
  // Init rules might be defined with
  or without a template
  // Check to make sure a template
  was used before trying
  // to access the parameters
  if (initRule instanceof
  DecisionTableTemplateInstanceRule)
  {
    DecisionTableTemplateIn
    stanceRule
    templateInstance =
    (DecisionTableTemplateI
    nstanceRule) initRule;

    RuleTemplate template =
    templateInstance.getRul
    eTemplate();
```

```
        List<Parameter>
        parameters =
        template.getParameters(
        );
        Iterator<Parameter>
        paramIterator =
        parameters.iterator();

        Parameter parameter =
        null;

        while
        (paramIterator.hasNext(
        )) {
        parameter =
        paramIterator.next();

         out.println("Parameter
        Name: " +
        parameter.getName());
        out.println("Parameter
        Value: "
        +
        templateInstance.getPar
        ameterValue(parameter
        .getName()));
        }
      }
    }
    // For the rest of the decision table, start at
    the root and
    // recursively work through the different case
    edges and
    // actions
    TreeBlock treeBlock =
    decisionTable.getTreeBlock();
    TreeNode treeNode = treeBlock.getRootNode();

    printDecisionTableConditionsAndActions(treeNode
    , 0);
    out.println("");
    return out.toString();
  }
  /*Method to recursively work through the case edges and print
  out the conditions and actions.
*/
  static private void printDecisionTableConditionsAndActions(
    TreeNode treeNode, int indent)
  {
    out.print("<table border=\"1\">");
    if (treeNode instanceof ConditionNode)
    {
      // Get the case edges for the
      current TreeNode
      // and for each case edge print out
      the conditions
      ConditionNode conditionNode =
      (ConditionNode) treeNode;

      List<CaseEdge> caseEdges =
      conditionNode.getCaseEdges();
      Iterator<CaseEdge> caseEdgeIterator
      = caseEdges.iterator();

      CaseEdge caseEdge = null;

      while (caseEdgeIterator.hasNext())
```

```
{
  out.print("<tr>");
  // If this is the start
  of the conditions for the
  // condition node,
  print out the condition term
  if (indent == 0)
  {
  out.print("<td>");

  TreeConditionTermDefinition
  termDefinition =
  conditionNode
  .getTermDefinition();

  out.print(termDefinitio
  n.getUserPresentation()
  );
  out.print("</td>");
  indent++;
  } else {
  // After the condition
  term has been printed
  for a
  // case edge skip for
  the rest of the case
  edges
  out.print("<td></td>");
  }

  caseEdge =
  caseEdgeIterator.next()
  ;

  out.print("<td>");

  // Check if the
  caseEdge is defined by
  a template
  if
  (caseEdge.getValueDefin
  ition() != null)
  {
  TemplateInstanceExpress
  ion templateInstance =
  caseEdge
  .getValueTemplateInstan
  ce();

  out.println(templateIns
  tance.getExpandedUserPr
  esentation());

  TreeConditionValueDefin
  ition valueDef =
  caseEdge
  .getValueDefinition();

  out.println(valueDef.ge
  tUserPresentation());

  Template template =
  templateInstance.getTem
  plate();

  // Get the parameters
  for the template
```

```
definition and
// print out the
parameter names and
values
List<Parameter>
parameters =
template.getParameters(
);
Iterator<Parameter>
paramIterator =
parameters.iterator();

List<ParameterValue>
parameterValues =
templateInstance
.getParameterValues();
Iterator<ParameterValue
> paramValues =
parameterValues
.iterator();

Parameter parameter =
null;
ParameterValue
parameterValue = null;

while
(paramIterator.hasNext(
) &&
paramValues.hasNext())
{
parameter =
paramIterator.next();
parameterValue =
paramValues.next();

out.println("Parameter
Name: " +
parameter.getName());
out.println("Parameter
Value: "
  +
parameterValue.getValue
());
}
}

out.print("</td><td>");
// Print the child node
for the caseEdge
printDecisionTableCondi
tionsAndActions(caseEdg
e.getChildNode(),
0);

out.print("</td></tr>")
;
}


// Add Otherwise condition if it
exists
TreeNode otherwise =
conditionNode.getOtherwiseCase();

if (otherwise != null)
{
```

```
    out.print("<tr><td></td>
    <td>Otherwise</td><td>
    ");
    // Print the Otherwise
    ConditionNode
    printDecisionTableCondi
    tionsAndActions(otherwi
    se, 0);
    out.print("</td></td>")
    ;
  }
  out.print("</table>");
} else {
  // ActionNode has been found and
  different logic is needed
  // to print out the TreeActions
  ActionNode actionNode =
  (ActionNode) treeNode;
  List<TreeAction> treeActions =
  actionNode.getTreeActions();

  Iterator<TreeAction>
  treeActionIterator =
  treeActions.iterator();

  TreeAction treeAction = null;

  // The ActionNode can contain
  multiple TreeActions to
  // print out
  while
  (treeActionIterator.hasNext())
  {
    out.print("<tr>");
    treeAction =
    treeActionIterator.next
    ();

    TreeActionTermDefinitio
    n treeActionTerm =
    treeAction
    .getTermDefinition();

    if (indent == 0) {
    out.print("<td>");
    out.print(treeActionTer
    m.getUserPresentation()
    );
    out.print("</td>");
    }
    out.print("<td>");
    TemplateInstanceExpress
    ion templateInstance =
    treeAction
    .getValueTemplateInstan
    ce();

    // Check that a
    template was specified
    for
    // the TreeAction
    before working with the
    // parameter name and
    values
    if (templateInstance !=
    null) {
    out.println(templateIns
```

```
tance.getExpandedUserPr
esentation());

Template template =
templateInstance.getTem
plate();

List<Parameter>
parameters =
template.getParameters(
);

Iterator<Parameter>
paramIterator =
parameters.iterator();

List<ParameterValue>
parameterValues =
templateInstance
.getParameterValues();
Iterator<ParameterValue
> paramValues =
parameterValues
.iterator();

Parameter parameter =
null;
ParameterValue
parameterValue = null;

while
(paramIterator.hasNext(
) &&
paramValues.hasNext())
{
{parameter =
paramIterator.next();
parameterValue =
paramValues.next();

out.println(" Parameter
Name: " +
parameter.getName());
out.println(" Parameter
Value: "
+
parameterValue.getValue
());

}
} else
{
// If a template was
not used, the only item
that is
// available is the
UserPresentation if it
was
// specified when the
rule was created
out.print(treeAction.ge
tValueUserPresentation(
));
}

out.print("</td></tr>")
;
```

```
      }
      out.print("</table>");
    }
  }
  /*
    Method to print out a rule set
  */
public static String printRuleSet(BusinessRule
ruleArtifact)
{
   out.clear();
   out.printlnBold("Rule Set");
   RuleSet ruleSet = (RuleSet) ruleArtifact;
   out.println("Name: " + ruleSet.getName());
   out.println("Namespace: " +
   ruleSet.getTargetNameSpace());

   // The rules in a rule set are contained in a
   rule block
   RuleBlock ruleBlock =
   ruleSet.getFirstRuleBlock();

   Iterator<RuleSetRule> ruleIterator =
   ruleBlock.iterator();

   RuleSetRule rule = null;

   // Iterate through the rules in the rule block.
   while (ruleIterator.hasNext())
   {
     rule = ruleIterator.next();
     out.printBold("Rule: ");
     out.println(rule.getName());
     out.println("Display Name: " +
     rule.getDisplayName());
     out.println("Description: " +
     rule.getDescription());
     // The expanded user presentation
     will automatically populate the
     // presentation with the parameter
     values and can be used for
     // display if the rule was defined
     with a template. If no
     // template was defined the
     expanded user presentation
     // is the same as the regular
     presentation.
     out.println("Expanded User
     Presentation: "
       +
     rule.getExpandedUserPre
     sentation());
     // The regular user presentation
     will have placeholders in the
     // string where the parameter can
     be substituted if the rule
     // was defined with a template. If
     the rule was not defined with
     // a template, the user
     presentation will only be a string
     // without placeholders. The
     placeholders are of a format of {n}
     // where n is the index (zerobased)
     of the parameter in the
     // template. This value can be used
     to create an interface for
     // editing where there are fields
```

```
        with the parameter values
        // available for editing
        out.println("User Presentation: " +
        rule.getUserPresentation());

        // Check if the rule was defined
        with a template
        if (rule instanceof
        RuleSetTemplateInstanceRule) {
          RuleSetTemplateInstance
          Rule templateInstance =
          (RuleSetTemplateInstanc
          eRule) rule;

          RuleSetRuleTemplate
          template =
          templateInstance
          .getRuleSetRuleTemplate
          ();

          List<Parameter>
          parameters =
          template.getParameters(
          );
          Iterator<Parameter>
          paramIterator =
          parameters.iterator();

          Parameter parameter =
          null;

          // Retrieve all of the
          parameters and output
          the name and value
          while
          (paramIterator.hasNext(
          ))
          {
          parameter =
          paramIterator.next();

          out.println("Parameter
          Name: " +
          parameter.getName());
          out.println("Parameter
          Value: "
            +
          templateInstance.getPar
          ameterValue(
          parameter.getName()).ge
          tValue());
          }
        }
      }
      out.println("");
      return out.toString();
    }
}
```

Programmation de widgets

Business Space fournit des widgets qui sont activés pour divers produits IBM de gestion des processus métier. Toutefois, vous pouvez également créer vos propres widgets et les intégrer avec ceux d'IBM. Les références ci-après expliquent pourquoi créer des widgets et comment les créer, et fournit des références aux API que vous pouvez utiliser pour créer vos propres widgets.

Dans le centre de documentation de Business Space, consultez les rubriques suivantes :

- Présentation du développement de widgets
- Guide de programmation des widgets

Chapitre 4. Développement d'applications client pour les tâches et processus métier

Vous pouvez utiliser un outil de modélisation pour compiler et déployer des tâches et des processus métier. L'interaction avec ces processus et ces tâches se produit lors de l'exécution. Par exemple, un processus est lancé ou les tâches sont réclamées et effectuées. Vous pouvez utiliser Business Process Choreographer Explorer pour interagir avec des processus ou des tâches, ou vous pouvez utiliser les API de Business Process Choreographer afin de développer des clients personnalisés pour ces interactions.

task_context

Ces clients peuvent être des clients EJB (Enterprise JavaBeans™), des clients de service Web ou encore des clients Web exploitant les composants JSF (JavaServer Faces) de Business Process Choreographer Explorer. Ce dernier fournit des API EJB (Enterprise JavaBeans) et des interfaces pour les services Web pour vous permettre de développer ces clients. L'API EJB est accessible via n'importe quelle application Java, y compris une autre application EJB. Il est possible d'accéder aux interfaces des services Web à partir des environnements Java ou Microsoft® .Net.

Concepts associés

«Comparaison entre les interfaces de programmation visant à interagir avec les processus métier et les tâches manuelles», à la page 353
Des interfaces de programmation génériques EJB (Enterprise JavaBeans), JMS (Java Message Service), REST (Representational State Transfer Services) ainsi que des interfaces de programmation de services Web sont disponibles pour la création d'applications client interagissant avec des processus métier et des tâches manuelles. Chacune de ces interfaces présente des caractéristiques différentes.

«Requêtes portant sur les données des processus métier et des tâches», à la page 355
Les données d'instance des processus métier et des tâches manuelles à exécution longue sont stockées de façon persistante dans la base de données et accessibles par le biais de requêtes. En outre, il est possible d'accéder aux données des modèles de processus métier et de tâche manuelle grâce à une interface de requête.

Tâches associées

«Développement d'applications client EJB pour des processus métier et des tâches manuelles», à la page 438
Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches manuelles installées sur WebSphere Process Server.

«Développement d'applications client de services Web pour des processus métier et des tâches manuelles», à la page 507
Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches manuelles via des API de services Web Business Process Choreographer. Le processus de développement d'applications client comprend un certain nombre d'étapes obligatoires et facultatives, notamment la génération d'un proxy de service Web et l'ajout de règles de sécurité et de transaction dans l'application client.

«Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer», à la page 521
Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

«Développement d'applications Web pour les processus métier et tâches manuelles à l'aide de composants JSF», à la page 529
Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches manuelles à des applications Web.

«Développement des pages JSP pour les messages de tâche et de processus», à la page 554
Business Process Choreographer Explorer fournit des formulaires d'entrée et de sortie par défaut pour afficher et saisir les données métier. Vous pouvez utiliser des pages JSP pour créer des formulaires d'entrée et de sortie définis par l'utilisateur.

«Création de plug-in pour personnaliser les fonctionnalités des tâches manuelles», à la page 556
Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches manuelles. L'application des plug-in est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des plug-in personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

Comparaison entre les interfaces de programmation visant à interagir avec les processus métier et les tâches manuelles

Des interfaces de programmation génériques EJB (Enterprise JavaBeans), JMS (Java Message Service), REST (Representational State Transfer Services) ainsi que des interfaces de programmation de services Web sont disponibles pour la création d'applications client interagissant avec des processus métier et des tâches manuelles. Chacune de ces interfaces présente des caractéristiques différentes.

L'interface de programmation que vous choisissez dépend de plusieurs facteurs, dont la fonctionnalité devant être fournie par votre application client, le fait que vous disposez ou non d'une infrastructure de client final existante, ou encore que vous souhaitiez ou non traiter les flux de tâches manuelles. Pour faciliter la sélection de l'interface appropriée, le tableau suivant compare les caractéristiques des interfaces de programmation EJB, JMS, REST et de services Web.

	Interface EJB	**Interface de service Web**	**Interface de message JMS**	**Interface REST**
Fonctionnalité	Cette interface est disponible à la fois pour les processus métier et les tâches manuelles. Utilisez cette interface pour créer des clients fonctionnant de manière générique avec des processus et des tâches.	Cette interface est disponible à la fois pour les processus métier et les tâches manuelles. Utilisez cette interface pour créer des clients destinés à un ensemble connu de processus et de tâches.	Cette interface est disponible uniquement pour les processus métier. Utilisez cette interface pour créer des clients de messagerie destinés à un ensemble connu de processus.	Cette interface est disponible à la fois pour les processus métier et les tâches manuelles. Utilisez cette interface pour créer des clients de type Web 2.0 conçus pour un ensemble connu de processus et de tâches.
Traitement des données	Prend en charge le chargement de schémas d'artefacts distants pour accéder aux métadonnées des objets métier. Si l'application client EJB est exécutée dans la même cellule que le serveur WebSphere Process Server auquel elle est connectée, les schémas requis par les objets métier des processus et des tâches ne doivent pas nécessairement être disponibles au niveau du client et peuvent être chargés depuis le serveur via le chargeur d'artefacts distants RAL (Remote Artifact Loader). Le chargeur RAL peut également être appliqué à plusieurs cellules si l'application client s'exécute sur une installation serveur complète de WebSphere Process Server. Cependant, le chargeur RAL n'est pas utilisable dans une configuration inter-cellules dans laquelle l'application client s'exécute dans une installation client de WebSphere Process Server.	Les artefacts de schémas relatifs aux données d'entrée et de sortie, ainsi qu'aux variables, doivent être disponibles dans un format reconnu par le client.	Les artefacts de schémas relatifs aux données d'entrée et de sortie, ainsi qu'aux variables, doivent être disponibles dans un format reconnu par le client.	Les artefacts de schémas relatifs aux données d'entrée et de sortie, ainsi qu'aux variables, doivent être disponibles dans un format reconnu par le client.

	Interface EJB	Interface de service Web	Interface de message JMS	Interface REST
Environnement client	Une installation de WebSphere Process Server ou une installation client de WebSphere Process Server	Tout environnement d'exécution prenant en charge les appels de services Web, y compris les environnements Microsoft .NET.	Tout environnement d'exécution prenant en charge les clients JMS, y compris les modules SCA utilisant des importations JMS SCA.	Tout environnement d'exécution prenant en charge les clients REST.
Sécurité	Sécurité Java Platform, Enterprise Edition (Java EE)	Sécurité des services Web.	Sécurité Java Platform, Enterprise Edition (Java EE) pour l'installation WebSphere Process Server. Vous pouvez également sécuriser les files d'attente dans lesquelles l'application client JMS place les messages d'interface API, par exemple via les mécanismes de sécurité de WebSphere MQ.	Les applications client appelant les méthodes REST doivent utiliser un mécanisme d'authentification HTTP adapté.

Une opération peut être exposée par plusieurs protocoles. Tenez compte des remarques générales suivantes si vous utilisez la même opération dans différents protocoles.

- Dans les interfaces Services Web et REST, tous les identificateurs d'objet, tels que PIID, AIID et TKIID, sont représentés par un type chaîne (string). Seule l'interface API EJB attend un ID d'objet à type sécurisé.
- La surcharge d'opération est seulement utilisée pour les méthodes EJB et non pour les opérations WSDL. Dans certains cas, il existe plusieurs opérations WSDL distinctes ; dans d'autres cas, il n'en existe qu'une seule, qui autorise toutes les variantes de paramètres soit par omission (`minOccurs="0"`), soit par l'emploi de valeurs Null (`nillable="true"`).
- Dans certaines méthodes EJB, les espaces de noms XML et les noms locaux sont passés dans des paramètres distincts. La plupart des opérations WSDL utilisent le type XML Schema QName pour passer ces paramètres.
- Les interactions asynchrones avec les opérations de demande-réponse WSDL à exécution longue, telles que l'opération callWithReplyContext dans l'interface EJB ou l'opération callAsync dans l'interface WSDL, sont représentées par l'opération call dans l'interface JMS.
- L'interface EJB renvoie un ensemble d'objets d'API, qui exposent les méthodes d'accès get et set des champs contenus. Les interfaces Services Web et REST renvoient des documents de types complexes (XML ou JSON) au client.
- Certains services de Human Task Manager opérant sur des tâches manuelles sont également disponibles comme services Business Flow Manager opérant sur des activités qui appellent une tâche manuelle.

Tâches associées

Développement d'applications client EJB
Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches manuelles installées sur WebSphere Process Server.

Développement d'applications client d'API de services Web
Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches manuelles via des API de services Web Business Process Choreographer. Le processus de développement d'applications client comprend un certain nombre d'étapes obligatoires et facultatives, notamment la génération d'un proxy de service Web et l'ajout de règles de sécurité et de transaction dans l'application client.

Développement d'applications client JMS
Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

Requêtes portant sur les données des processus métier et des tâches

Les données d'instance des processus métier et des tâches manuelles à exécution longue sont stockées de façon persistante dans la base de données et accessibles par le biais de requêtes. En outre, il est possible d'accéder aux données des modèles de processus métier et de tâche manuelle grâce à une interface de requête.

Les interfaces EJB d'interrogation, l'API de requête et l'API de table de requête, sont disponibles avec Business Process Choreographer.

En fonction des clients qui accèdent aux données relatives aux processus ou aux tâches, vous pouvez utiliser une ou plusieurs de ces interfaces. Les API REST et Web Services (services Web) sont fournies dans Business Process Choreographer pour vous permettre d'interroger les données les listes de tâches et de processus. Cependant, pour des raisons de performances, si vous devez interroger des listes de processus et de tâches particulièrement volumineuses, utilisez l'API de table de requête EJB ou l'API de table de requête REST de Business Process Choreographer.

Concepts associés

«Comparaison des interfaces de programmation destinées à l'extraction de données de processus et de tâche»
Business Process Choreographer fournit deux interfaces de programmation (API) spécialisées dans l'extraction des données de processus et de table : l'API de table de requête et l'API de requête. Chacune de ces interfaces présente des caractéristiques différentes.

«Tables de requête dans Business Process Choreographer», à la page 358
Les tables de requête permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches manuelles et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requête fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requête sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

«API de requête EJB de Business Process Choreographer», à la page 419
Les méthodes query ou queryAll du service API vous permettent d'extraire des informations stockées relatives aux processus métier et aux tâches.

Comparaison des interfaces de programmation destinées à l'extraction de données de processus et de tâche

Business Process Choreographer fournit deux interfaces de programmation (API) spécialisées dans l'extraction des données de processus et de table : l'API de table de requête et l'API de requête. Chacune de ces interfaces présente des caractéristiques différentes.

L'interface choisie dépendra de plusieurs facteurs, dont la fonctionnalité attendue de votre application cliente, la disponibilité ou non d'une infrastructure de client pour utilisateur final existante et, bien sûr, le niveau de performances visé. Pour vous aider à faire le bon choix, le tableau suivant compare les caractéristiques des API de table de requête et de requête.

Caractéristique	API de table de requête	API de requête
Disponibilité	L'API de table de requête est disponible pour l'interface EJB de Business Flow Manager et pour l'interface de programmation REST.	L'API de requête est disponible pour les interfaces de programmation EJB, services Web, JMS et REST.
Méthodes dédiées à l'extraction de contenus	L'API fournit les méthodes suivantes : • queryEntities • queryEntityCount • queryRows • queryRowCount	L'API fournit les méthodes suivantes : • query • queryAll • queryProcessTemplates • queryTaskTemplates
Méthodes dédiées à l'extraction de métadonnées	L'API fournit les méthodes suivantes : • getQueryTableMetaData • findQueryTableMetaData	L'API fournit les méthodes suivantes : • QueryResultSet.getColumnType
Nom de la table de requête	Spécifie la table de requête sur laquelle l'API de table de requête est exécutée. Il n'est possible d'interroger qu'une seule table de requête à la fois. Par exemple, `queryEntities("CUST.TASKS", ...)`.	La clause SELECT spécifie les colonnes et les vues de base de données prédéfinies sur lesquelles la requête est exécutée. Cette spécification est similaire à une clause SQL SELECT. Par exemple, `query("TASK.TKIID, TASK.STATE, WORK_ITEM.REASON", ...)`.

Caractéristique	API de table de requête	API de requête
Clause SELECT et attributs sélectionnés	Utilisez les options de filtrage de l'API de table de requête pour spécifier les attributs que la requête doit renvoyer dans ses résultats. Comme la requête s'exécute sur une seule table de requête, les attributs sont identifiables sans équivoque par leurs noms respectifs.	Utilisez la clause SELECT pour spécifier les attributs. Un nom d'attribut est spécifié avec une syntaxe de la forme *nom_vue.nom_attribut*. Par exemple, pour rechercher des états de tâche, spécifiez TASK.STATE dans votre requête.
Clause WHERE et filtres	Avec l'API de table de requête, utilisez la propriété queryCondition pour filtrer encore plus les résultats de vos requêtes. Les tables de requête fournissent un contenu préfiltré si, dans leur définition, des filtres de table de requête principale, des filtres d'autorisation ou des filtres de table de requête ont été spécifiés.	Utilisez la clause WHERE pour filtrer le résultat de la requête.
Clause WHERE et critères de sélection	La clause WHERE de l'API de requête n'est pas nécessaire sous cette forme dans l'API de table de requête. Avec l'API de table de requête, utilisez la propriété queryCondition pour appliquer un filtrage additionnel. Les critères de sélection spécifiés dans la définition de table de requête sélectionnent une propriété particulière de la table de requête attachée. Ce comportement est obtenu en plus du filtrage exercé par la clause WHERE dans l'API de requête.	Les critères de sélection ne sont pas disponibles pour l'API de requête. Cependant, ils sont similaires à la partie de la clause WHERE qui définit, par exemple, le nom (colonne NAME) ou l'environnement local (colonne LOCALE) de la table ou vue QUERY_PROPERTY, TASK_CPROP ou TASK_DESC. Par exemple, une clause WHERE telle que `QUERY_PROPERTY.NAME='xyz'` revient au même que de spécifier `NAME='xyz'` comme critère de sélection dans la définition de la table de requête attachée QUERY_PROPERTY.
Eléments de travail et autorisation	Utilisez la table de requête WORK_ITEM pour accéder aux éléments de travail. Vous pouvez personnaliser l'emploi des éléments de travail dans la définition d'une table de requête au moment où vous la développez, ainsi que dans l'API de table de requête, en utilisant l'objet AuthorizationOptions ou AdminAuthorizationOptions. Par exemple, pour exclure les éléments de travail 'everybody' lors de l'interrogation de la table de requête TASK, spécifiez `WI.EVERYBODY=0` pour la propriété queryCondition, ou alors spécifiez `setUseEverbody(Boolean.FALSE)` sur l'objet AuthorizationOptions.	Utilisez la vue WORK_ITEM pour accéder aux éléments de travail. Les quatre types d'éléments de travail sont pris en compte pour les résultats d'une requête : 'everybody', 'individual', 'groups' et 'inherited'. Pour filtrer les éléments de travail d'après un type spécifique, personnalisez la clause WHERE. Par exemple, pour exclure les éléments de travail 'everybody', spécifiez `WORK_ITEM.EVERYBODY=0` dans la clause WHERE.
Paramètres	Vous pouvez utiliser des paramètres dans les filtres et les critères de sélection pour les tables de requête composites.	Les paramètres ne sont pas disponibles pour l'API de requête, sauf si des requêtes stockées sont utilisées.
Requêtes stockées et tables de requête	La différence entre une requête stockée et une table de requête est que la première est définie pour une requête particulière, tandis que la table de requête est définie pour un ensemble particulier de requêtes. Par exemple, la définition d'une table de requête n'autorise pas la spécification d'une clause ORDER BY, car les informations correspondantes ne sont généralement disponibles qu'au moment où la requête est exécutée.	Les requêtes stockées permettent d'interroger les données en utilisant des ensembles prédéfinis d'options.
Vues matérialisées	Les vues matérialisées ne sont pas disponibles pour l'API de table de requête.	Les vues matérialisées utilisent des technologies propres aux bases de données pour améliorer les performances des requêtes.
Tables personnalisées	Les tables de requête supplémentaires offrent la même fonctionnalité que les tables personnalisées.	Les tables personnalisées servent à inclure, dans les requêtes, des données qui sont externes au schéma de base de données de Business Process Choreographer.

Caractéristique	API de table de requête	API de requête
queryAll et options d'autorisation	La fonctionnalité queryAll est fournie par l'objet AdminAuthorizationOptions, lequel peut être passé à l'API de table de requête à la place de l'objet AuthorizationOptions. L'appelant doit faire partie du rôle BPESystemAdministrator, TaskSystemAdministrator, BPESystemMonitor ou TaskSystemMonitor.	La méthode queryAll permet aux utilisateurs ayant le rôle Java EE BPESystemAdministrator d'obtenir tous les objets dans les résultats d'une requête sans être limités par les éléments de travail d'un utilisateur ou d'un groupe particulier.
Internationalisation	Lors de l'utilisation de tables de requête, il est possible de choisir la langue dans laquelle doivent être présentés les noms et les descriptions des tables en question et de leurs attributs.	Les noms des colonnes des vues sélectionnées, tels qu'ils apparaissent dans la base de données ou tels qu'ils sont spécifiés dans la clause de sélection, sont renvoyés.

Tables de requête dans Business Process Choreographer

Les tables de requête permettent d'interroger les données des listes de tâches et des listes de processus contenues dans le schéma de base de données de Business Process Choreographer. Les requêtes peuvent porter sur les données des tâches manuelles et celles des processus métier gérés par Business Process Choreographer, mais aussi sur les données de processus métier externes. Les tables de requête fournissent une abstraction des données de Business Process Choreographer qui peut être exploitée par les applications clientes, lesquelles deviennent ainsi indépendantes de l'implémentation proprement dite des tables. Les définitions des tables de requête sont déployées dans des conteneurs Business Process Choreographer et sont accessibles au moyen d'une API spécifique, appelée API de table de requête.

Il existe trois types de tables de requête :

- Tables de requête prédéfinies
- Tables de requête supplémentaires
- Tables de requête composites

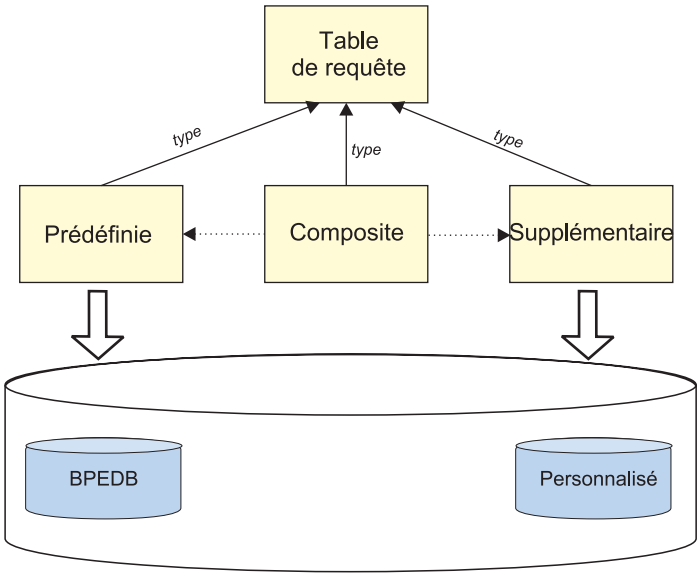


Figure 72. Tables de requête dans Business Process Choreographer

Les tables de requête sont représentées au moyen de modèles similaires dans l'environnement d'exécution et vous pouvez les interroger à l'aide de l'API de table

de requête. Tandis que les tables de requête prédéfinies et supplémentaires pointent directement sur des tables ou des vues dans la base de données, les tables composites sont constituées de parties de ces données, représentées sous la forme d'une table unique.

Les tables de requête étendent les possibilités offertes par les vues de base de données prédéfinies et les interfaces de requête existantes de Business Process Choreographer, et :

- Elles sont optimisées pour exécuter des requêtes de liste de processus et de tâches en utilisant des modèles d'accès eux-mêmes optimisés pour délivrer les meilleures performances.
- Elles simplifient et consolident l'accès aux informations nécessaires.
- Elles permettent de configurer très précisément les options d'autorisation et de filtrage.

Il est possible de personnaliser des tables de requête, par exemple, configurer une table de requête pour qu'elle contienne uniquement les tâches ou les instances de processus correspondant à un scénario particulier. Il est également recommandé d'utiliser des tables de requête lorsque les performances sont primordiales, par exemple dans le cas de requêtes concernant une liste de processus ou de tâches volumineuse.

L'outil Query Table Builder est fourni sous forme de plug-in Eclipse pour vous permettre :

- de développer des tables de requête composites et supplémentaires ;
- d'importer et d'exporter des définitions de tables de requête au format XML.

Vous pouvez télécharger l'outil Query Table Builder à partir du site des SupportPacs de WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

Concepts associés

«Tables de requête prédéfinies», à la page 361
Les tables de requête prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requête, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE. Ces tables de requête prédéfinies améliorent les fonctionnalités et les performances des vues de base de données car elles sont optimisées pour exécuter des requêtes portant sur les listes de processus et de tâches.

«Tables de requête supplémentaires», à la page 364
Les tables de requête supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requête un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requête supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requête prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches manuelles.

«tables de requête composites», à la page 366
Dans Business Process Choreographer les tables de requête composites n'ont pas de représentation spécifique de données dans la base de données ; il s'agit d'une combinaison des données de tables de requête prédéfinies et supplémentaires prédéfinies. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

«Développement des tables de requête», à la page 374
Dans Business Process Choreographer, le développement des tables de requête supplémentaires et des tables de requête composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requête prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

«Filtres et critères de sélection des tables de requête», à la page 378
Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requête, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requête.

«Autorisation pour les tables de requête», à la page 383
Vous pouvez utiliser une autorisation par instance, une autorisation par rôle ou aucune autorisation lorsque vous exécutez des requêtes ou des tables de requête.

«Types d'attribut pour les tables de requête», à la page 388
Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requête, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

«Requêtes sur des tables de requête», à la page 394
L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

«Requêtes sur des tables de requête pour l'extraction de métadonnées», à la page 408
L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête. Des méthodes sont disponibles pour permettre l'extraction de métadonnées des tables de requête.

«Internationalisation pour les métadonnées des tables de requête», à la page 410
L'internationalisation est prise en charge pour les métadonnées des tables de requête.

«Tables de requête et performances des requêtes», à la page 412
Les tables de requête offrent un nouveau modèle de programmation propre, conçu pour le développement d'applications clientes qui extraient des listes de tâches manuelles et de processus métier dans Business Process Choreographer. Le recours aux tables de requête améliore les performances. Des informations sont fournies sur les paramètres d'API de table de requête et les autres facteurs ayant un impact sur les performances.

Tâches associées

«Création de tables de requête pour Business Space», à la page 416
Dans l'outil Query Table Builder, vous pouvez utiliser la définition de table de requête composite contenant des propriétés prédéfinies en vue de créer des tables de requête pour Business Space.

«Création de tables de requête pour Business Process Choreographer Explorer», à la page 417
Vous pouvez utiliser des tables de requête à la place de l'API query des EJB pour améliorer les performances de Business Process Choreographer Explorer. Pour créer les tables de requête, utilisez Query Table Builder.

Tables de requête prédéfinies

Les tables de requête prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requête, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE. Ces tables de requête prédéfinies améliorent les fonctionnalités et les performances des vues de base de données car elles sont optimisées pour exécuter des requêtes portant sur les listes de processus et de tâches.

Les tables de requête prédéfinies peuvent être interrogées directement à l'aide de l'API de table de requête. Lorsque vous accédez aux tables en utilisant cette API, vous avez un plus grand choix d'options de configuration qu'avec l'API de requête.

Propriétés

Les tables de requête prédéfinies ont les propriétés suivantes :

Tableau 25. Propriétés des tables de requête prédéfinies

Propriété	Description
Nom	Le nom de la table de requête est le nom, en majuscules, de l'une des vues prédéfinies de la base de données, par exemple, TASK.
Attributs	Les attributs des tables de requête prédéfinies définissent les éléments d'information disponibles pour les requêtes. Il s'agit des noms en majuscules des colonnes spécifiées par les vues prédéfinies de la base de données. Les attributs sont définis avec un nom et un type. Le type est l'un des suivants : • **Boolean** : une valeur booléenne • **Decimal** : un nombre en virgule flottante • **ID** : un ID d'objet, tel que le TKIID de la table de requête TASK • **Number** : un entier, court (type short) ou long (type long) • **String** : une chaîne • **Timestamp** : un horodatage
Autorisation	Les tables de requête prédéfinies utilisent soit l'autorisation par instance, soit l'autorisation par rôle. • Pour les tables de requête prédéfinies qui contiennent des données d'instance, l'autorisation par instance est obligatoire. Cela signifie que la requête renvoie seulement les objets avec des éléments de travail destinés à l'utilisateur qui exécute la requête. Cependant, en utilisant l'objet AdminAuthorizationOptions, vous pouvez limiter cette vérification à un simple contrôle de l'existence d'un élément de travail pour tout utilisateur. L'utilisateur doit disposer du rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé, ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé pour ces requêtes. • Les tables de requête prédéfinies avec des données de modèle requièrent une autorisation par rôle, ce qui signifie que seuls les utilisateurs faisant partie du rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé, peuvent accéder au contenu de ces tables de requête.

Tables de requête prédéfinies avec des données d'instance

Le tableau suivant présente les tables de requête prédéfinies qui contiennent des données d'instance. Ces tables de requête se caractérisent par ce qui suit :

- Elles peuvent être utilisées comme table de requête principale d'une table composite.
- Elles utilisent l'autorisation par instance si elles sont interrogées directement. La technique utilisée à cet effet est une jointure (SQL-) avec la vue qui stocke les informations d'autorisation, c'est-à-dire la vue ou la table de requête prédéfinie WORK_ITEM.
- Elles contiennent des données d'instance ; par exemple, celles d'instances de tâche ou d'instances de processus.

Tableau 26. Tables de requête prédéfinies contenant des données d'instance

Données d'instance	Nom de la table de requête
Informations sur les activités d'une instance de processus.	ACTIVITY
	ACTIVITY_ATTRIBUTE
	ACTIVITY_SERVICE
Informations sur les escalades appartenant aux tâches manuelles.	ESCALATION
	ESCALATION_CPROP
	ESCALATION_DESC
Informations sur les instances de processus.	PROCESS_ATTRIBUTE
	PROCESS_INSTANCE
	QUERY_PROPERTY
Informations sur les tâches manuelles.	TASK
	TASK_CPROP
	TASK_DESC

La table de requête WORK_ITEM contient aussi des données d'instance, mais elle n'est pas utilisable comme table de requête principale ni comme table de requête attachée. Les données d'élément de travail sont disponibles implicitement lors de l'interrogation des tables de requête qui utilisent l'autorisation par instance. Autrement dit, lorsque vous interrogez une table de requête utilisant l'autorisation par instance, vous pouvez utiliser comme critères les attributs de la table WORK_ITEM, même s'ils ne sont pas explicitement spécifiés par la table que vous interrogez.

Tables de requête prédéfinies avec des données de modèle

Pour les tables de requête prédéfinies qui contiennent des données de modèle, l'autorisation par rôle est obligatoire. Elles ne peuvent être interrogées que par les administrateurs, à l'aide de l'objet AdminAuthorizationOptions.

Le tableau suivant présente les tables de requête prédéfinies qui contiennent des données de modèle. Ces tables de requête se caractérisent par ce qui suit :

- Elles peuvent être utilisées comme table de requête principale d'une table composite.
- Elles utilisent l'autorisation par rôle si elles sont interrogées directement. Cela signifie que l'appelant utilisant la méthode de requête d'API doit faire partie du rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé, ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé, et AdminAuthorizationOptions doit être utilisé.
- Elles contiennent des données de modèle ; par exemple, celles des modèles de tâche ou des modèles de processus.

Tableau 27. Tables de requête prédéfinies contenant des données de modèle

Données de modèle	Nom de la table de requête
Informations sur les composants d'application.	APPLICATION_COMP

Tableau 27. Tables de requête prédéfinies contenant des données de modèle (suite)

Données de modèle	Nom de la table de requête
Informations sur les modèles d'escalade.	ESC_TEMPL
	ESC_TEMPL_CPROP
	ESC_TEMPL_DESC
Informations sur les modèles de processus.	PROCESS_TEMPLATE
	PROCESS_TEMPL_ATTR
Informations sur les modèles de tâche.	TASK_TEMPL
	TASK_TEMPL_CPROP
	TASK_TEMPL_DESC

Concepts associés

«Tables de requête supplémentaires»
Les tables de requête supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requête un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requête supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requête prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches manuelles.

«tables de requête composites», à la page 366
Dans Business Process Choreographer les tables de requête composites n'ont pas de représentation spécifique de données dans la base de données ; il s'agit d'une combinaison des données de tables de requête prédéfinies et supplémentaires prédéfinies. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

«Développement des tables de requête», à la page 374
Dans Business Process Choreographer, le développement des tables de requête supplémentaires et des tables de requête composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requête prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

«Requêtes sur des tables de requête», à la page 394
L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

«Autorisation pour les tables de requête», à la page 383
Vous pouvez utiliser une autorisation par instance, une autorisation par rôle ou aucune autorisation lorsque vous exécutez des requêtes ou des tables de requête.

Tables de requête supplémentaires

Les tables de requête supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requête un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requête supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requête prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches manuelles.

Les tables de requête supplémentaires sont associées à des tables ou des vues dans la base de données de Business Process Choreographer. Il s'agit de tables de requête contenant des données métier gérées par des applications client. Les tables de requête supplémentaires fournissent des informations dans une table composite qui vient en complément du contenu d'une table de requête prédéfinie.

Les tables de requête supplémentaires ont les propriétés suivantes :

Tableau 28. Propriétés des tables de requête supplémentaires

Propriété	Description
Nom	Le nom d'une table de requête doit être unique au sein d'une même installation Business Process Choreographer. A l'exécution de la requête, c'est ce nom qui est utilisé pour identifier la table de requête interrogée. Une table de requête est identifiée de manière unique par son nom, qui est de la forme *préfixe.nom*. La longueur de *préfixe.nom* ne doit pas dépasser 28 caractères. Le préfixe `'IBM'` est réservé et ne doit pas être utilisé ; par exemple, `'COMPANY.BUS_DATA'` est un nom correct. N'indiquez pas de chiffre à la fin du nom de la table. Si une table est utilisée plusieurs fois dans une requête, son nom est complété par un numéro compris entre 0 et 9. Par exemple, CUSTOM_VIEW0, CUSTOM_VIEW1, etc. Si le nom d'origine de la table contient déjà un chiffre, celui-ci est supprimé par Business Process Choreographer, ce qui génère une exception QueryUnknownTableException.
Nom de base de données	Le nom de la table ou vue associée dans la base de données. Seules des lettres majuscules sont acceptées.
Schéma de base de données	Le schéma de la table ou vue associée dans la base de données. Seules des lettres majuscules sont acceptées. Ce schéma doit être différent de celui de la base de données de Business Process Choreographer. Cependant, la table ou la vue doit être accessible avec la même source de données JDBC que celle qui est utilisée pour accéder à la base de données de Business Process Choreographer.
Attributs	Les attributs des tables de requête supplémentaires définissent les éléments d'information disponibles pour les requêtes. Ces attributs doivent avoir le même nom que les colonnes auxquelles ils correspondent dans la table ou la vue de base de données. Les attributs sont définis avec un nom et un type. Le nom est défini en majuscules. Le type est l'un des suivants : • **Boolean** : une valeur booléenne • **Decimal** : un nombre en virgule flottante • **ID** : un ID d'objet d'une longueur de 16 octets, tel que le TKIID de la table de requête TASK • **Number** : un entier, court (type short) ou long (type long) • **String** : une chaîne • **Timestamp** : un horodatage
Jointures	Des jointures doivent être définies sur les tables supplémentaires si elles sont attachées à une table dite "principale" pour former des tables de requête composites. Une jointure définit quels attributs sont utilisés pour corréler les informations de la table de requête supplémentaire avec celles de la table de requête principale. Lorsqu'une jointure est définie, l'attribut source et l'attribut cible doivent être du même type.

Tableau 28. Propriétés des tables de requête supplémentaires (suite)

Propriété	Description
Autorisation	Aucun contrôle d'autorisation n'est spécifié pour les tables de requête supplémentaires ; par conséquent, tous les utilisateurs authentifiés peuvent voir leur contenu.

Concepts associés

«Tables de requête prédéfinies», à la page 361
Les tables de requête prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requête, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE. Ces tables de requête prédéfinies améliorent les fonctionnalités et les performances des vues de base de données car elles sont optimisées pour exécuter des requêtes portant sur les listes de processus et de tâches.

«tables de requête composites»
Dans Business Process Choreographer les tables de requête composites n'ont pas de représentation spécifique de données dans la base de données ; il s'agit d'une combinaison des données de tables de requête prédéfinies et supplémentaires prédéfinies. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

«Développement des tables de requête», à la page 374
Dans Business Process Choreographer, le développement des tables de requête supplémentaires et des tables de requête composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requête prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

«Requêtes sur des tables de requête», à la page 394
L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

«Autorisation pour les tables de requête», à la page 383
Vous pouvez utiliser une autorisation par instance, une autorisation par rôle ou aucune autorisation lorsque vous exécutez des requêtes ou des tables de requête.

tables de requête composites

Dans Business Process Choreographer les tables de requête composites n'ont pas de représentation spécifique de données dans la base de données ; il s'agit d'une combinaison des données de tables de requête prédéfinies et supplémentaires prédéfinies. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Les tables de requête composites sont conçues par les développeurs de logiciels client et autorisent une configuration fine des filtres et des options d'autorisation, permettant ainsi d'optimiser l'accès aux données lors de l'exécution des requêtes. Elles sont créées au moyen du code SQL, qui est optimisé pour les requêtes sur les listes de tâches et de processus.

Il est recommandé d'utiliser des tables de requête composites dans les scénarios de production à la place des API de requête de Business Process Choreographer standard car elles fournissent une abstraction de l'implémentation proprement dite des requêtes et permettent donc de les optimiser.

De plus, vous pouvez modifier les tables de requête composite au moment de l'exécution sans avoir à redéployer le client qui accède à la table de requête.

La figure suivante offre une vue d'ensemble du contenu des tables de requête composites :

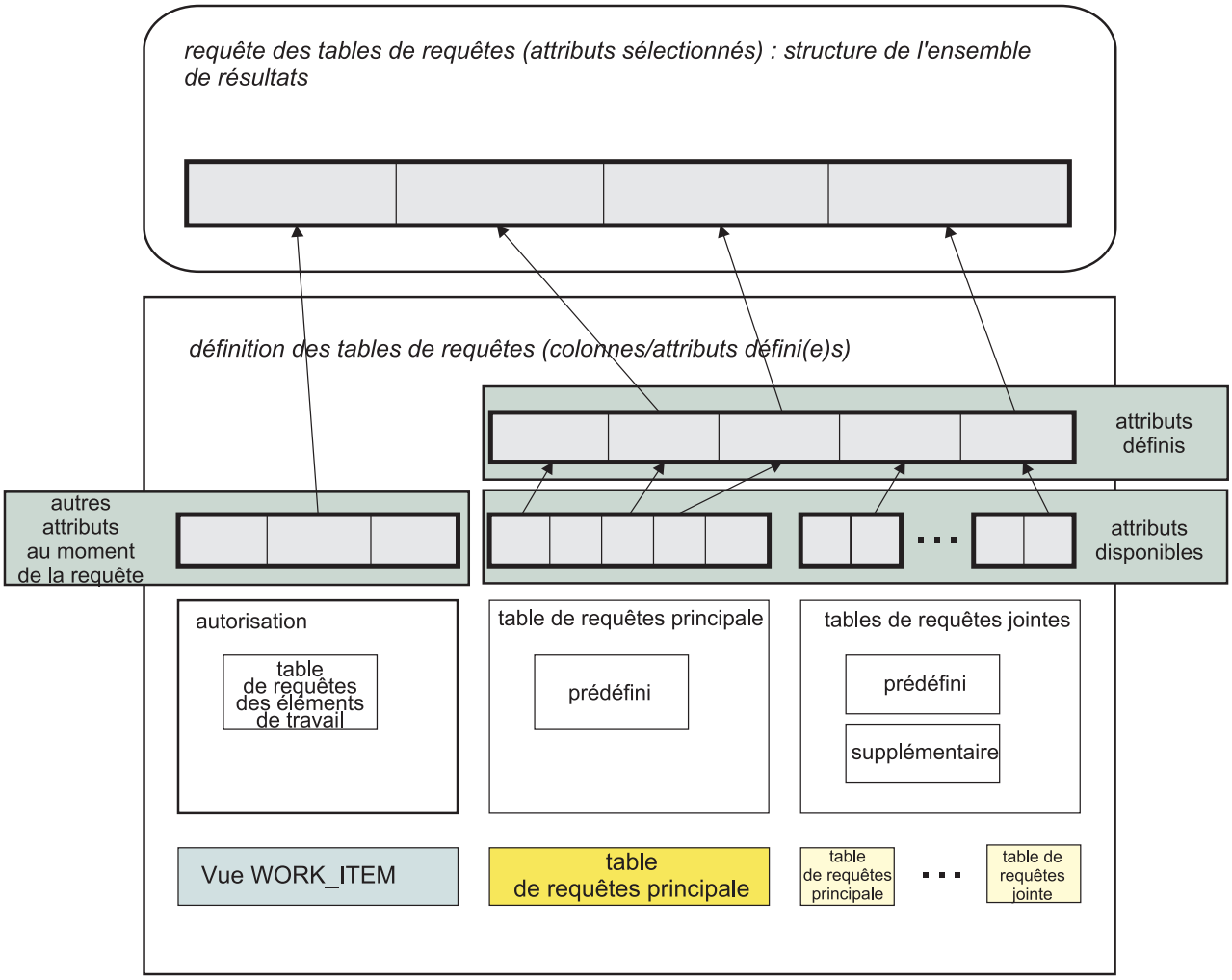


Figure 73. Contenu des tables de requête composites

Toutes les tables de requête composites sont définies avec une table de requête principale et zéro, une ou plusieurs tables de requête attachées.

La table de requête principale :

- Constitue l'information principale contenue dans une table de requête composite.
- Doit être l'une des tables de requête prédéfinies.
- Identifie de manière unique, par sa clé primaire, chaque objet contenu dans la table de requête composites. Par exemple, dans le cas de la table de requête prédéfinie TASK, il s'agit de l'ID de tâche TKIID.
- Contrôle l'accès au contenu d'une table de requête en utilisant les éléments de travail qui figurent dans la table de requête WORK_ITEM, si l'autorisation par instance est utilisée.
- Détermine la liste des objets retournés comme lignes d'une table lorsque la table composite est interrogée.

Les tables de requête attachées :

- Peuvent être des tables de requête prédéfinies ou supplémentaires déjà déployées sur le système.
- Ont pour objectif de fournir des informations complémentaires de celles qui sont fournies par la table de requête principale. Par exemple, si TASK est la table de requête principale, la description de chaque tâche fournie dans la table de requête TASK_DESC peut être ajoutée au contenu de la table de requête composite.

En général, la table de requête principale est choisie en fonction de l'objectif de la table de requête composite.

- Si la table composite doit décrire une liste de tâches, la table de requête TASK sera désignée comme table principale.
- Si la table composite doit décrire une liste de processus, la table de requête PROCESS_INSTANCE sera désignée comme table principale.
- Les listes d'activités sont obtenues en utilisant ACTIVITY comme table de requête principale.
- Les listes d'escalades de tâches manuelles sont obtenues en utilisant ESCALATION comme table de requête principale.

Relation entre la table de requête principale et les tables de requête associées

La table de requête associée et la table de requête principale doivent avoir une relation un à un ou un à zéro. Si le type de relation un à un ou un à zéro n'est pas respecté, une exception d'exécution (RuntimeException) se produira à l'exécution de la requête.

La corrélation entre table principale et chaque table attachée est réalisée par un attribut de jointure qui est défini sur la table attachée. Pour les tables de requête prédéfinies, cet attribut de jointure ne peut pas être changé, car il décrit la relation entre les données dans les différentes tables de requête de Process Choreographer. L'attribut de jointure est généralement suffisant pour maintenir la relation un à un ou un à zéro. Par exemple, l'attribut CONTAINMENT_CTX_ID est utilisé sur la table de requête TASK pour attacher les données de l'instance de processus associée, laquelle est identifiée par l'attribut PIID dans la table de requête PROCESS_INSTANCE.

Lorsqu'une relation un à plusieurs existe, vous devez indiquer un critère supplémentaire, appelé critère de sélection, dans l'outil Query Table Builder lorsque vous définissez la table de requête. Par exemple, il peut s'agir de l'expression `"LOCALE='fr_FR'"`. Une tâche peut avoir plusieurs descriptions identifiées chacune par un code d'environnement local différent.

Exemple 1 :

La figure suivante illustre l'emploi de critères de sélection spécifiés sur les tables de requête attachées :

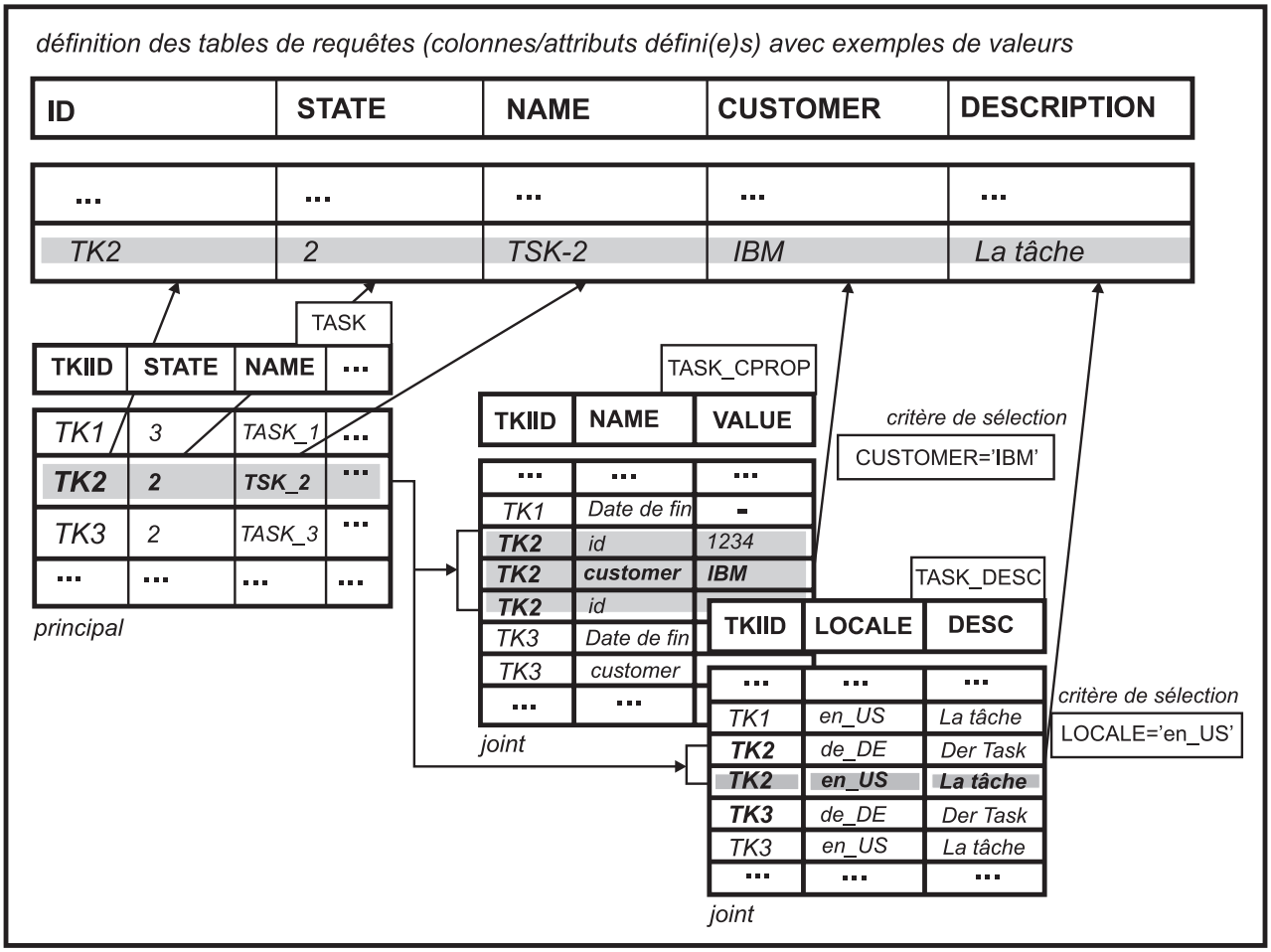


Figure 74. Table de requête composite avec critères de sélection

La table de requête composite contient les attributs ID, STATE, NAME, CUSTOMER et DESCRIPTION.

- Les attributs ID (le TKIID), STATE et NAME sont fournis par la table de requête principale TASK.
- CUSTOMER est une propriété personnalisée définie sur la table TASK. Les propriétés personnalisées sont stockées dans la table de requête TASK_CPROP. Pour une tâche particulière, une propriété personnalisée est identifiée de manière unique par son nom. Ce point est reflété dans le critère de sélection `"CUSTOMER='IBM'"`.
- DESCRIPTION est la description de la tâche ; elle est stockée dans la table de requête TASK_DESC. La description de tâche associée à une instance de tâche particulière est identifiée de manière unique par son environnement local. Ce point est reflété dans le critère de sélection `"LOCALE='en_US'"`.

Exemple 2 :

L'objectif de cet exemple est d'illustrer la relation entre la table principale et la table attachée en utilisant TASK comme table principale et TASK_DESC comme table attachée. Lorsque vous définissez votre table de requête composite, l'attribut LOCALE de la table de requête TASK_DESC doit être indiqué pour garantir l'existence d'une relation un à un ou un à zéro entre la table de requête principale

et la table de requête associée. Le tableau suivant montre des exemples de contenus d'une table composite, avec un critère de sélection valide spécifié pour la table attachée TASK_DESC.

Tableau 29. Contenus valides d'une table de requête composite

Table principale TASK	**Table attachée TASK_DESC**	
NAME	LOCALE	DESCRIPTION
task_one	en_US	This is a description.
task_two	en_US	This is a description.
...	...	...

Le tableau suivant présente des contenus hypothétiques qui ne sont pas valides (en **gras**) si le critère de sélection n'est pas défini correctement, ce qui signifie que la relation un à un ou un à zéro n'est pas respectée.

Tableau 30. Contenus non valides d'une table de requête composite

TASK (table de requête principale)	**TASK_DESC (table de requête attachée)**	
NAME	LOCALE	DESCRIPTION
task_one	en_US	This is a description.
task_one	**de_DE**	**Das ist eine Beschreibung.**
...	...	...

Propriétés

Les tables de requête composites ont les propriétés suivantes :

Tableau 31. Propriétés des tables de requête composites

Propriété	**Description**
Nom	Le nom d'une table de requête doit être unique au sein d'une même installation Business Process Choreographer. A l'exécution de la requête, c'est ce nom qui est utilisé pour identifier la table de requête interrogée. Une table de requête est identifiée de manière unique par son nom, qui est de la forme *préfixe.nom* dans le cas d'une table de requête composite. La longueur de *préfixe.nom* ne doit pas dépasser 28 caractères. Le préfixe 'IBM' est réservé et ne doit pas être utilisé ; par exemple, 'COMPANY.TODO_TASK_LIST' est un nom correct. N'indiquez pas de chiffre à la fin du nom de la table. Si une table est utilisée plusieurs fois dans une requête, son nom est complété par un numéro compris entre 0 et 9. Par exemple, CUSTOM_VIEW0, CUSTOM_VIEW1, etc. Si le nom d'origine de la table contient déjà un chiffre, celui-ci est supprimé par Business Process Choreographer, ce qui génère une exception QueryUnknownTableException.

Tableau 31. Propriétés des tables de requête composites (suite)

Propriété	Description
Attributs	Les attributs des tables de requête composites définissent les éléments d'information disponibles pour les requêtes. Les attributs sont définis avec un nom en majuscules. Leur type est hérité de celui de l'attribut auquel ils font référence, qui est l'un des suivants : • **Boolean** : une valeur booléenne • **Decimal** : un nombre en virgule flottante • **ID** : un ID d'objet, tel que le TKIID de la table de requête TASK • **Number** : un entier, court (type short) ou long (type long) • **String** : une chaîne • **Timestamp** : un horodatage Les attributs d'une table de requête composite sont définis via une référence aux attributs de la table principale ou des tables attachées. Les attributs des tables de requête composites héritent des types et des constantes des attributs auxquels ils font référence. Outre les attributs qui font partie de la définition de la table de requête, les données d'élément de travail peuvent être interrogées à l'exécution. Les conditions à remplir sont que la table de requête principale contienne des données d'instance, (c'est le cas des tables de requête TASK et PROCESS_INSTANCE) et que l'autorisation par instance soit utilisée sur la table de requête composite. Par exemple, la requête peut être définie de manière à renvoyer uniquement les tâches manuelles pour lesquelles l'utilisateur est un propriétaire potentiel.

Tableau 31. Propriétés des tables de requête composites (suite)

Propriété	Description
Autorisation	Chaque table de requête composite définit si l'autorisation par instance ou par rôle est utilisée lorsque des requêtes sont exécutées dessus (ou s'il n'y a pas de contrôle d'autorisation). Si l'autorisation par instance est définie, la requête renvoie seulement les objets avec des éléments de travail destinés à l'utilisateur qui exécute la requête. Cependant, en utilisant l'objet AdminAuthorizationOptions, vous pouvez limiter cette vérification à un simple contrôle de l'existence d'un élément de travail pour tout utilisateur. L'utilisateur doit faire partie du rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé, pour ces requêtes, et AdminAuthorizationOptions doit être transmis à l'API de table de requête. Si une autorisation par rôle est définie, l'utilisateur doit faire partie du rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé, pour ces requêtes, et AdminAuthorizationOptions doit être transmis à l'API de table de requête. Si aucun contrôle d'autorisation n'est défini, l'exécution de la requête a lieu sans vérification préalable de l'existence des éléments de travail des objets associés dans la table de requête. Tous les utilisateurs authentifiés peuvent voir le contenu de la table de requête. L'autorisation par instance peut être définie si la table de requête principale contient des données d'instance ; l'autorisation par rôle peut être définie si la table de requête principale contient des données de modèle. L'absence de contrôle d'autorisation peut être définie sur les tables de requête composites, quelle que soit leur table principale.

Filtres

Les filtres servent à limiter le nombre d'objets ou de lignes contenus dans une table de requête composite.

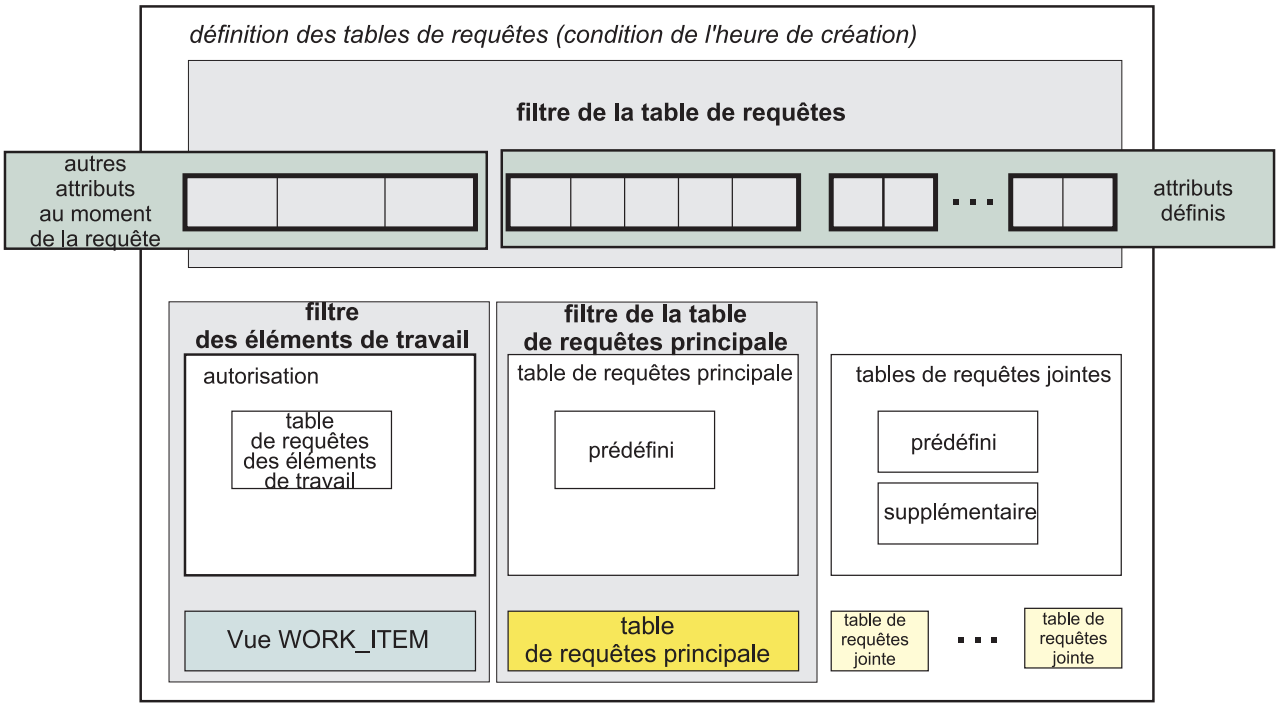


Figure 75. Filtres dans les tables de requête composites

Pendant le développement d'une table de requête composite, des filtres peuvent être définis et appliqués :

- à la table de requête principale, comme filtre de la table de requête principale ;
- à la table de requête WORK_ITEM, laquelle est implicitement disponible et assure le contrôle d'accès (autorisation) si la table de requête principale contient des données d'instance. Ce filtre est appelé filtre d'autorisation ; il est disponible uniquement si la table composite est configurée pour utiliser l'autorisation par instance.
- à la table de requête composite, comme filtre de table de requête.

Les filtres sont définis durant le développement des tables de requête. Par exemple, avec une table de requête composite dont la table principale est TASK, il est possible de filtrer les tâches de manière à exclure toutes celles qui ne sont pas à l'état prêt (en utilisant l'expression `"STATE=STATE_READY"` comme filtre de la table de requête principale).

Autorisation

L'autorisation d'accès au contenu d'une table de requête composite est similaire à celle qui réglemente l'accès à la table utilisée comme table principale. La différence est que la table composite peut être configurée pour être plus restrictive.

- Si l'autorisation par instance est utilisée, les données contenues dans la table composite sont vérifiées par rapport aux éléments de travail dans la table de requête WORK_ITEM. Cette vérification s'effectue par rapport à la table de requête principale. Les éléments de travail 'everybody', 'individual', 'group' et 'inherited' sont utilisés pour la vérification, selon la configuration de la table de requête composite. Si des éléments de travail 'inherited' sont spécifié, les objets ayant comme parent une instance de processus (par exemple, une tâche manuelle participante) et qui sont liés à un élément de travail 'everybody',

'individual' ou 'group' (en fonction de la configuration), sont présents dans la table de requête composite. En général, les éléments de travail 'inherited' ne sont utiles qu'aux administrateurs.

- Les tables composites dont la table principale contient des données de modèle ne doivent pas être configurées pour utiliser l'autorisation par instance. Si l'autorisation par rôle est utilisée, les requêtes peuvent être exécutées uniquement par les utilisateurs qui font partie du rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé, ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé, et l'objet AdminAuthorizationOptions doit être utilisé.

Concepts associés

«Tables de requête prédéfinies», à la page 361
Les tables de requête prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requête, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE. Ces tables de requête prédéfinies améliorent les fonctionnalités et les performances des vues de base de données car elles sont optimisées pour exécuter des requêtes portant sur les listes de processus et de tâches.

«Tables de requête supplémentaires», à la page 364
Les tables de requête supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requête un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requête supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requête prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches manuelles.

Développement des tables de requête

Dans Business Process Choreographer, le développement des tables de requête supplémentaires et des tables de requête composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requête prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

L'outil Query Table Builder, disponible sous forme de plug-in Eclipse, peut être téléchargé sur le site des SupportPacs WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

L'utilisation de tables de requête a une incidence sur le développement et le déploiement des applications. Les étapes suivantes décrivent les rôles mis en jeu lorsque vous concevez et développez une application Business Process Choreographer utilisant des tables de requête.

Tableau 32. Etapes de développement de tables de requête

Etape	Rôle	Description
1. Analyse	Analyste métier, développeur de client	Analysez les besoins de l'application cliente et déterminez quelles tables de requête sont nécessaires. Posez-vous les questions suivantes : • Combien de listes de tâches ou de processus faut-il fournir à l'utilisateur ? Y a-t-il des listes de tâches ou de processus qui pourraient partager la même table de requête ? • Quel type d'autorisation est utilisé ? Par instance, par rôle ou aucun ? • Existe-t-il, dans le système, d'autres tables de requête déjà prêtes à l'emploi qui pourraient être réutilisées ? • Les tables de requête doivent-elles fournir le contenu en plusieurs langues ? Si oui, les critères de sélection sur les tables de requête attachées doivent être `LOCALE=$LOCALE`.
2. Développement des tables de requête	Développeur de client, analyste métier	Développez les tables de requête à utiliser dans l'application cliente. Efforcez-vous de spécifier leur définition de manière à optimiser les performances obtenues avec les requêtes qu'elles fournissent.
3. Déploiement des tables de requête	Administrateur	Les tables de requête doivent être déployées dans l'environnement d'exécution avant de pouvoir être utilisées. Cette étape est réalisée au moyen de la commande wsadmin manageQueryTable.py.
4. Requêtes sur les tables de requête	Développeur de client	La dernière étape du développement consiste à exécuter des requêtes sur les tables. Le développeur de l'application cliente doit connaître le nom de la table à interroger et ses attributs.

L'exemple de code ci-dessous utilise l'API de table de requête pour interroger une table de requête. Pour des raisons de simplification, les exemples 1 et 2 fournis interrogent la table de requête prédéfinie TASK. Les exemples 3 et 4 interrogent une table de requête composite et supposent que celle-ci a été préalablement déployée sur le système. Dans le cadre du développement d'applications, il est recommandé d'utiliser des tables de requête composites au lieu d'interroger directement les tables de requête prédéfinies.

Exemple 1

```
// permet d'obtenir le contexte d'affectation de nom et de rechercher l'interface
// EJB home Flow Manager Enterprise ; l'EJB home Business Flow
// doit être mis en mémoire cache pour des raisons de performance ;
// nous supposons également qu'il existe une référence Enterprise JavaBeans
// à l'EJB local de Business Flow Manager
Context ctx = new InitialContext();
LocalBusinessFlowManagerHome home =
 (LocalBusinessFlowManagerHome)
 ctx.lookup("java:comp/env/ejb/BFM");
```

```
// si l'EJB Human Task Manager est utilisé, entrez :
// LocalHumanTaskManagerHome home =
//   (LocalHumanTaskManagerHome) ctx.lookup("java:comp/env/ejb/HTM");
// en supposant qu'une référence EJB à l'EJB Human Task Manager
// a été définie

// crée le module de remplacement de Business Flow Manager côté client
LocalBusinessFlowManager bfm = home.create();
// si l'EJB Human Task Manager est utilisé, entrez :
// LocalHumanTaskManager htm = home.create();
// l'EJB Human Task Manager fournit les mêmes méthodes
// que l'EJB Business Flow Manager
// ************************************************
// ****************** exemple 1 *******************
// ************************************************

// exécute une requête sur la table de requête prédéfinie
// TASK ; il s'agit ici d'une simple liste Mes tâches
EntityResultSet ers = null;
ers = bfm.queryEntities("TASK", null, null, null);

// imprime le résultat dans STDOUT
EntityInfo entityInfo = ers.getEntityInfo();
List attList = entityInfo.getAttributeInfo();
int attSize = attList.size();

Iterator iter = ers.getEntities().iterator();
while( iter.hasNext() ) {
 System.out.print("Entity: ");
 Entity entity = (Entity) iter.next();
 for (int i = attSize - 1; i >= 0; i--) {
  AttributeInfo ai = (AttributeInfo) attList.get(i);
  System.out.print(
    entity.getAttributeValue(ai.getName()));
 }
 System.out.println();
}
```

Exemple 2

```
// ************************************************
// ****************** exemple 2 *******************
// ************************************************

// identique à l'exemple 1, mais utilise des requêtes
// sur des lignes
RowResultSet rrs = null;
rrs = bfm.queryRows("TASK", null, null, null);

attList = rrs.getAttributeInfo();
attSize = attList.size();

// imprime le résultat dans STDOUT
while (rrs.next()) {
 System.out.print("Row: ");
 for (int i = attSize - 1; i >= 0; i--) {
  AttributeInfo ai = (AttributeInfo) attList.get(i);
  System.out.print(
    rrs.getAttributeValue(ai.getName()));
 }
 System.out.println();
}
```

Exemple 3

```
   // ************************************************
   // ****************** exemple 3 *******************
   // ************************************************

   // exécute une requête sur une table de requête composite
   // préalablement déployée sur le système ;
   // on suppose que le nom est COMPANY.TASK_LIST
   ers = bfm.queryEntities(
     "COMPANY.TASK_LIST", null, null, null);
^
   //  imprime le résultat dans STDOUT ...
```

Exemple 4

```
   // ************************************************
   // ****************** exemple 4 *******************
   // ************************************************

   // requête sur la même table de requête que dans l'exemple 3,
   // mais utilise des options personnalisées
   FilterOptions fo = new FilterOptions();

   // renvoie uniquement les objets à l'état Prêt
   fo.setQueryCondition("STATE=STATE_READY");

   // trie les objets en fonction de leur ID
   fo.setSortAttributes("ID");

   // limite à 50 le nombre d'entités
   fo.setThreshold(50);

   // récupère uniquement un sous-ensemble des attributs définis
   // au niveau de la table de requête
   fo.setSelectedAttributes("ID, STATE, DESCRIPTION");

   AuthorizationOptions ao = new AuthorizationOptions();

   // ne renvoie pas les objets que tous les utilisateurs sont
   // autorisés à voir
   ao.setEverybodyUsed(Boolean.FALSE);

   ers = bfm.queryEntities(
     "COMPANY.TASK_LIST", fo, ao, null);

   //  imprime le résultat dans STDOUT ...
```

Concepts associés

«Requêtes sur des tables de requête», à la page 394
L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

«Filtres et critères de sélection des tables de requête»
Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requête, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requête.

«Tables de requête prédéfinies», à la page 361
Les tables de requête prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requête, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE. Ces tables de requête prédéfinies améliorent les fonctionnalités et les performances des vues de base de données car elles sont optimisées pour exécuter des requêtes portant sur les listes de processus et de tâches.

«Tables de requête supplémentaires», à la page 364
Les tables de requête supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requête un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requête supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requête prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches manuelles.

«tables de requête composites», à la page 366
Dans Business Process Choreographer les tables de requête composites n'ont pas de représentation spécifique de données dans la base de données ; il s'agit d'une combinaison des données de tables de requête prédéfinies et supplémentaires prédéfinies. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

Filtres et critères de sélection des tables de requête

Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requête, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requête.

Vous trouverez des informations sur l'installation de l'outil Query Table Builder sur le site des SupportPacs de WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

Attributs

Les attributs utilisés dans une expression sont ceux des tables de requête. Selon l'endroit où se situe l'expression, différents attributs sont disponibles. Le développeur d'applications clientes ne peut utiliser des expressions que dans les filtres de requête qu'il passe à l'API de table de requête. Le développeur de tables de requête composites a quant à lui la possibilité d'utiliser des expressions en des endroits plus variés. Le tableau suivant décrit les attributs disponibles et en quels endroits ils sont utilisables.

Tableau 33. Les attributs des tables de requête et leur utilisation dans les expressions

Où	Expression	Attributs disponibles
API de table de requête	Filtre de requête	• Tous les attributs définis dans la table de requête. • Si l'autorisation par instance est utilisée, tous les attributs définis dans la table de requête WORK_ITEM, préfixés avec `'WI.'` . Exemples : • `STATE=STATE_READY`, si la table de requête contient un attribut STATE et si une constante STATE_READY est définie pour cet attribut • `STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER`, si la table de requête contient un attribut STATE et si elle utilise l'autorisation par instance
Table de requête composite	Filtre de table de requête	
	Filtre de la table de requête principale	• Tous les attributs définis pour la table de requête principale. Exemple : • `STATE=STATE_READY`, si la table de requête contient un attribut STATE et si une constante STATE_READY est définie pour cet attribut
	Filtre d'autorisation	• Tous les attributs définis dans la table de requête prédéfinie WORK_ITEM, préfixés avec `'WI.'` . Exemple : • `WI.REASON=REASON_POTENTIAL_OWNER`
	Critère de sélection	• Tous les attributs définis dans la table de requête attachée et associée. Exemple : • `LOCALE='en_US'`, si la table de requête attachée contient un attribut LOCALE, ce qui est le cas de la table TASK_DESC

La figure suivante montre les différents endroits où les filtres et critères de sélection peuvent être utilisés dans des expressions ; elle inclut également des exemples :

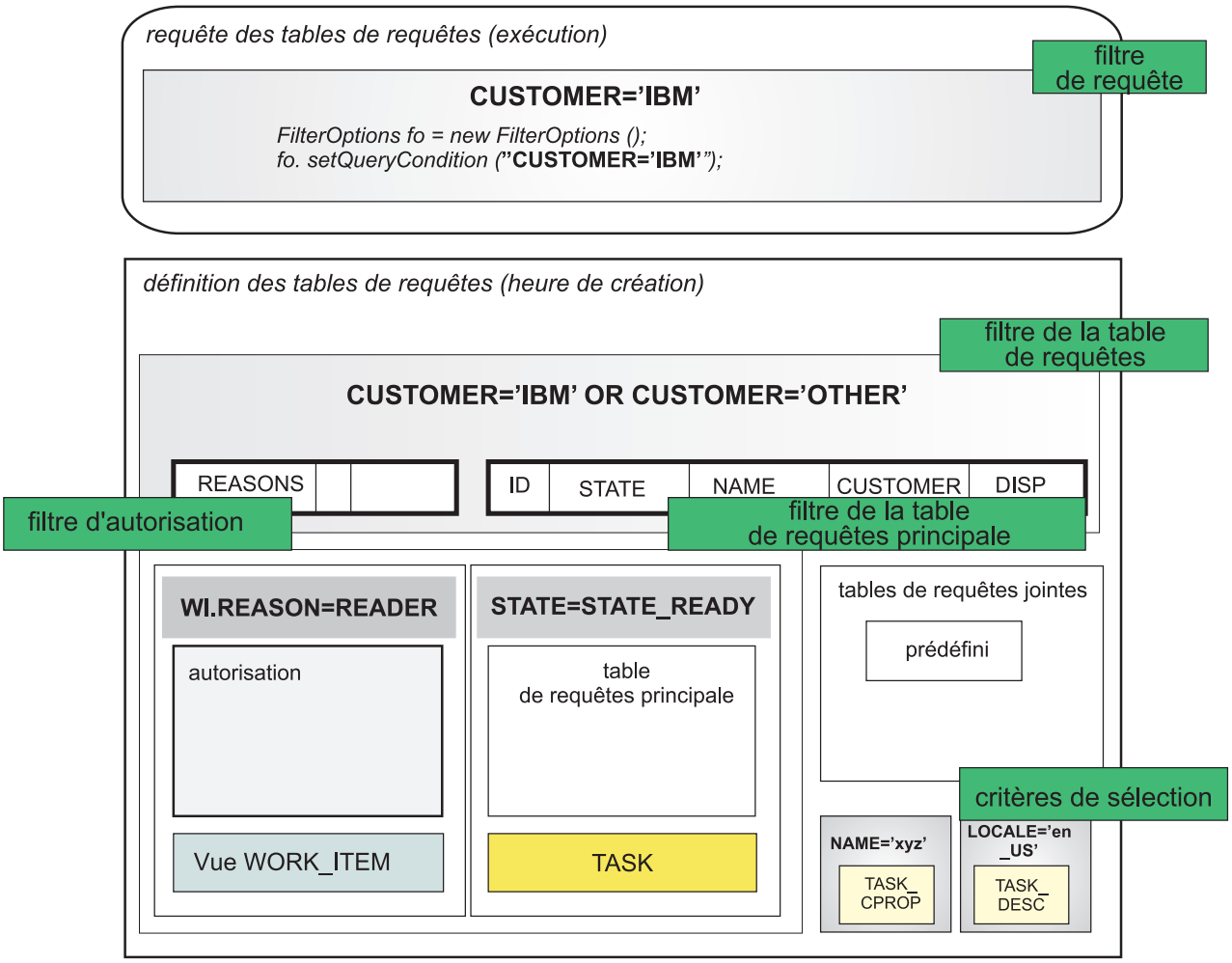


Figure 76. Filtres et critères de sélection dans des expressions

Expressions

Les expressions ont la syntaxe suivante :

```
expression  ::=  attribut op_binaire valeur |
                 attribut op_unaire |
                 attribut op_liste liste |
                 (expression) |
                 expression AND expression |
                 expression> OR expression
```

Les règles suivantes s'appliquent :

- L'opérateur `AND` est évalué avant l'opérateur `OR`. Les sous-expressions peuvent être reliées par des opérateurs `AND` et `OR`.
- Les expressions peuvent être groupées au moyen de parenthèses, qui doivent être appariées.

Exemples :

- `STATE = STATE_READY`
- `NAME IS NOT NULL`
- `STATE IN (2, 5, STATE_FINISHED)`
- `((PRIORITY=1) OR (WI.REASON=2)) AND (STATE=2)`

Une expression est prise en compte et évaluée dans une portée précise, qui détermine les attributs valides pour cette expression. Les critères de sélection ou les filtres de requête sont pris en compte et évalués dans la portée de la table de requête sur laquelle la requête est exécutée.

L'exemple suivant s'applique à une requête exécutée sur la table de requête prédéfinie TASK :

```
'(STATE=STATE_READY AND WI.REASON=REASON_POTENTIAL_OWNER)
OR (WI.REASON=REASON_OWNER)'
```

Opérateurs binaires

Les opérateurs binaires suivants sont disponibles :

op_binaire ::= = | < | > | <> | <= | >= | LIKE | NOT LIKE

Les règles suivantes s'appliquent :

- L'opérande côté gauche d'un opérateur binaire doit faire référence à un attribut d'une table de requête.
- L'opérande côté droit d'un opérateur binaire doit être une valeur littérale, une valeur constante ou un paramètre.
- Les opérateurs LIKE et NOT LIKE sont utilisables uniquement avec les attributs du type d'attribut STRING.
- Les opérandes côtés gauche et droit doivent avoir des types d'attribut compatibles.
- Les paramètres utilisateur doivent être compatibles avec le type d'attribut de l'attribut côté gauche.

Exemples :

- `STATE > 2`
- `NAME LIKE 'start%'`
- `STATE <> PARAM(theState)`

Opérateurs unaires

Les opérateurs unaires suivants sont disponibles :

op_unaire ::= IS NULL | IS NOT NULL

Les règles suivantes s'appliquent :

- L'opérande côté gauche d'un opérateur unaire doit faire référence à un attribut d'une table de requête. Les attributs valides dépendent de l'endroit où le filtre ou le critère de sélection est utilisé.
- Tous les attributs peuvent être testés pour déterminer s'ils sont NULL ; par exemple : CUSTOMER IS NOT NULL.

Exemple :

```
DESCRIPTION IS NOT NULL
```

Opérateurs de liste

Les opérateurs de liste suivants sont disponibles :

op_liste ::= IN | NOT IN

Les règles suivantes s'appliquent :

- L'opérande côté droit d'un opérateur de liste ne doit pas être remplacé par un paramètre utilisateur.
- Des paramètres utilisateurs peuvent apparaître dans la liste de l'opérande côté droit.

Exemple :

```
STATE IN (STATE_READY, STATE_RUNNING, PARAM(st), 1)
```

Les listes sont représentées comme suit :

liste ::= *valeur* [, *liste*]

Les règles suivantes s'appliquent :

- L'opérande côté droit d'un opérateur de liste ne doit pas être remplacé par un paramètre utilisateur.
- Des paramètres utilisateurs peuvent apparaître dans la liste de l'opérande côté droit.

Exemples :

- `(2, 5, 8)`
- `(STATE_READY, STATE_CLAIMED)`

Valeurs

Dans les expressions, une valeur peut être l'une des suivantes :

- **Constante** : une valeur constante, définie pour l'attribut concerné d'une table de requête prédéfinie. Par exemple, `STATE_READY` est une constante définie pour l'attribut STATE de la table prédéfinie TASK.
- **Littéral** : toute valeur codée en dur.
- **Paramètre** : un paramètre est remplacé par une valeur spécifique lors de l'exécution de la requête.

Des **constantes** sont disponibles pour certains attributs des tables de requête prédéfinies. Pour obtenir des informations sur les constantes disponibles pour les attributs de tables de requête prédéfinies, reportez-vous à la description des vues prédéfinies. Seules les constantes qui définissent des valeurs entières sont exposées dans les tables de requête. Il est également possible d'utiliser, à la place des constantes, les valeurs littérales correspondantes ou des paramètres.

Exemples :

- `STATE_READY`, constante propre à l'attribut STATE de la table de requête TASK, peut être utilisée dans un filtre pour déterminer si la tâche est à l'état prêt.
- `REASON_POTENTIAL_OWNER`, constante propre à l'attribut REASON de la table de requête WORK_ITEM, peut être utilisée dans un filtre pour tester si l'utilisateur qui exécute la requête sur une table de requête est un propriétaire potentiel.
- Le filtre de requête `STATE=STATE_READY` est identique à `STATE=2`, si la requête est exécutée sur la table de requête TASK.

Des **littéraux** peuvent aussi être utilisés dans les expressions. Une syntaxe spéciale doit être employée pour les horodatages et les ID.

Exemples :

- `STATE=1`

- `NAME='theName'`
- `CREATED > TS ('2008-11-26 T12:00:00')`
- `TKTID=ID('_TKT:801a011e.9d57c52.ab886df6.1fcc0000')`

L'utilisation de **paramètres** dans les expressions permet de donner un caractère dynamique aux tables de requête composites. Il existe des paramètres utilisateur et des paramètres système :

- Un paramètre utilisateur est spécifié avec le format PARAM (*nom*). Ce paramètre (sa valeur) doit être fourni au moment où la requête est exécutée. Il est passé à l'API de table de requête sous forme d'instance de la classe com.ibm.bpe.api.Parameter.
- Les paramètres système sont ainsi appelés parce que leurs valeurs sont fournies par l'environnement d'exécution (runtime) de table de requête, sans être spécifiées lors de l'exécution de la requête. Les paramètres système disponibles sont `$USER` et `$LOCALE`.
  - `$USER` est une chaîne identifiant l'utilisateur qui exécute la requête.
  - `$LOCALE` est une chaîne spécifiant le code standard de l'environnement local en vigueur au moment où la requête est exécutée. `'fr_FR'` est un exemple de valeur utilisable pour le paramètre `$LOCALE`.

Vous pouvez spécifier un paramètre dans le critère de sélection d'une table de requête attachée afin de sélectionner un environnement local spécifique. Par exemple, si TASK est la table principale d'une table de requête composite et si TASK_DESC est l'une des tables attachées, vous pouvez faire en sorte d'obtenir la description de la tâche dans une langue particulière. Voici quelques exemples d'utilisation de paramètres :

- `STATE=PARAM(theState)`
- `LOCALE=$LOCALE`
- `OWNER=$USER`

Concepts associés

«Développement des tables de requête», à la page 374
Dans Business Process Choreographer, le développement des tables de requête supplémentaires et des tables de requête composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requête prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

«Requêtes sur des tables de requête», à la page 394
L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Tâches associées

«Création de tables de requête pour Business Process Choreographer Explorer», à la page 417
Vous pouvez utiliser des tables de requête à la place de l'API query des EJB pour améliorer les performances de Business Process Choreographer Explorer. Pour créer les tables de requête, utilisez Query Table Builder.

Autorisation pour les tables de requête

Vous pouvez utiliser une autorisation par instance, une autorisation par rôle ou aucune autorisation lorsque vous exécutez des requêtes ou des tables de requête.

Le type d'autorisation est défini dans la table de requête.

- L'autorisation par instance indique que les objets dans la table de requête sont autorisés moyennant l'utilisation d'un élément de travail. A cet effet, un contrôle de l'existence d'un élément de travail adéquat est effectué.
- L'autorisation par rôle se fonde sur les rôles Java EE. Cela indique que l'appelant utilisant la méthode de requête d'API doit faire du partie rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé, ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé pour afficher le contenu de la table de requête. Ce type d'autorisation est disponible pour les tables de requête prédéfinies avec des données de modèle et pour les tables composites dont la table principale contient des données de modèle. Les objets contenus dans ces tables de requête n'ont pas d'élément de travail associé.
- Si aucun contrôle d'autorisation n'est spécifié, tous les utilisateurs authentifiés peuvent voir l'intégralité du contenu de la table de requête, après application des filtres.

Les types d'autorisation appliqués aux tables de requête prédéfinies, ainsi que les types d'autorisation qu'il est possible de configurer sur les tables composites et sur les tables supplémentaires, sont décrits dans le tableau suivant.

Tableau 34. Types d'autorisation pour les tables de requête

Table de requête	**Autorisation par instance**	**Autorisation par rôle**	**Absence de contrôle d'autorisation**
Prédéfinie	Requise pour les tables de requête prédéfinies qui contiennent des données d'instance.	Requise pour les tables de requête prédéfinies qui contiennent des données de modèle.	N/A
Composite	Peut être désactivée, auquel cas il n'y a pas de contrôle d'autorisation et les contraintes de sécurité sont passées outre. Cela signifie que chaque utilisateur authentifié peut utiliser la table de requête pour extraire des données, qu'il soit ou non autorisé à accéder aux objets correspondants. Les tables composites dont la table principale contient des données de modèle ne doivent pas être configurées pour utiliser l'autorisation par instance.	Peut être désactivée, par exemple pour les tables composites dont la table principale contient des données de modèle. Dans ce cas, il n'y a pas de contrôle d'autorisation et les contraintes de sécurité sont passées outre. Cela signifie que chaque utilisateur authentifié peut utiliser la table de requête pour extraire des données, qu'il soit ou non autorisé à accéder aux objets correspondants. Les tables composites dont la table principale contient des données d'instance ne doivent pas être configurées pour utiliser l'autorisation par rôle.	Tous les utilisateurs authentifiés peuvent voir l'intégralité du contenu de la table de requête, après application des filtres.
Supplémentaire	Les tables de requête supplémentaires ne doivent pas être configurées pour utiliser l'autorisation par instance, car Business Process Choreographer ne gère pas ces tables lui-même et il ne dispose donc pas d'informations sur les autorisations d'accès à leur contenu.	Les tables de requête supplémentaires ne doivent pas être configurées pour utiliser l'autorisation par rôle.	Tous les utilisateurs authentifiés peuvent voir l'intégralité du contenu de la table de requête, après application des filtres.

La figure suivante offre une vue d'ensemble des options disponibles pour les types d'autorisation, en fonction du type de table de requête. Elle illustre aussi les différents comportements obtenus avec l'API de table de requête et ses options d'autorisation.

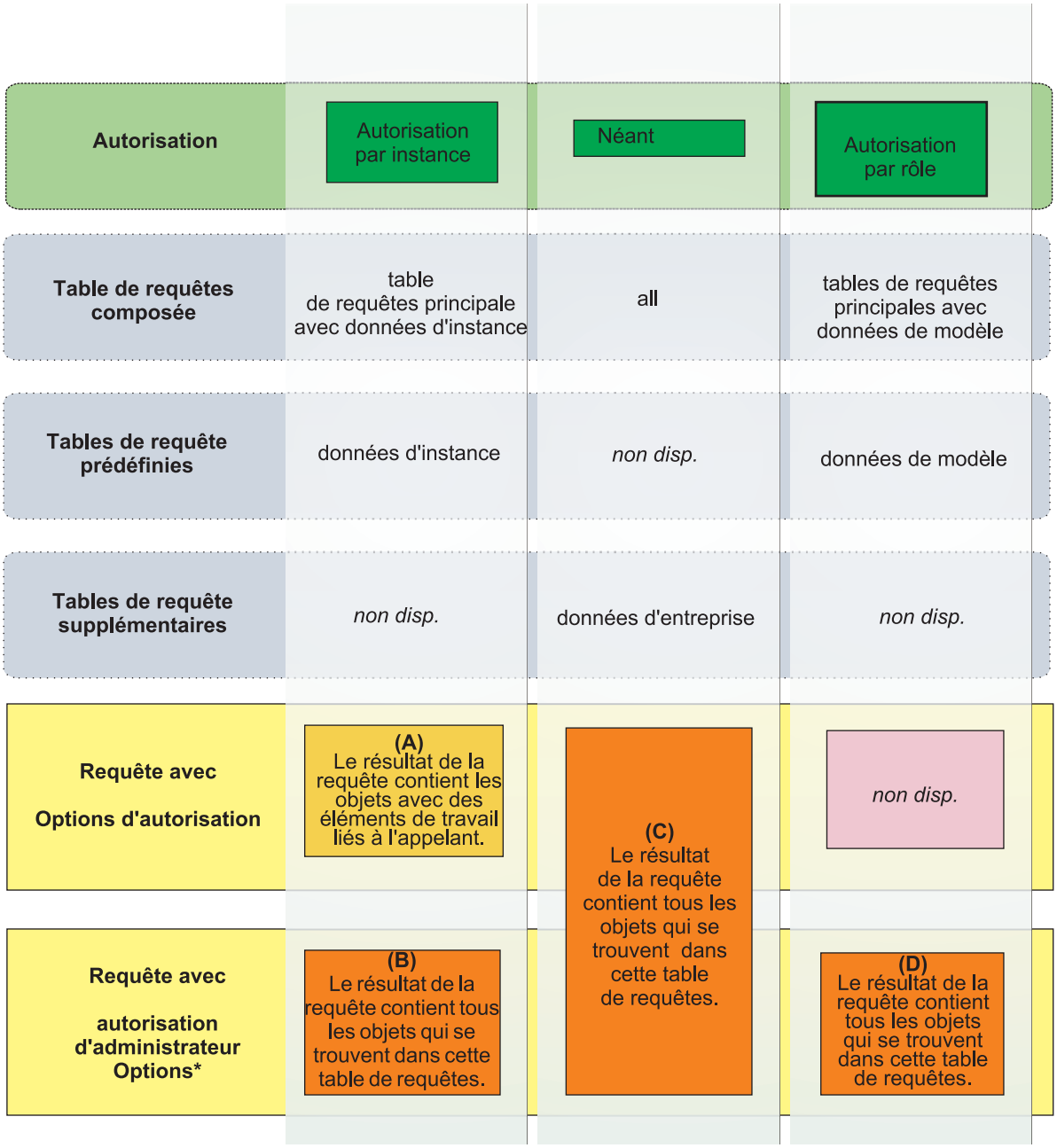


Figure 77. Autorisation par instance pour les tables de requête

*) Si le paramètre onBehalfUser est défini explicitement, le comportement (A) s'applique

L'autorisation par instance utilisant les éléments de travail et s'exerçant sur les objets renvoyés dans les résultats d'une requête dépend du paramètre d'autorisation qui est passé à l'API de table de requête, mais aussi de la valeur (true ou false) de l'option Autorisation par instance spécifiée dans la définition de la table de requête interrogée.

- (A) Les requêtes ciblant des tables prédéfinies ou composites et utilisant l'objet AuthorizationOptions renvoient les entités qui concordent avec un élément de travail approprié à l'utilisateur désigné. C'est également le cas si l'objet AdminAuthorizationOptions est utilisé et si le paramètre onBehalfUser est défini avec une valeur explicite (l'ID d'un utilisateur). Les clients standard, qui présentent des listes de tâches ou de processus aux utilisateurs, emploient généralement cette combinaison de types de table de requête et de paramètres d'API.
- (B) Le contenu intégral d'une table de requête est constitué des entités qui ont un élément de travail correspondant, tel que configuré avec l'autorisation par instance de la table en question. L'autorisation par instance considère quatre types d'éléments de travail : 'everybody', 'individual', 'group' et 'inherited'. L'appelant utilisant la méthode de requête d'API doit faire partie du rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé, ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé. Cette combinaison de types de tables de requête et de paramètres d'API est destinée aux scénarios d'administration, dans lesquels la liste complète des tâches ou processus disponibles doit pouvoir être affichée ou parcourue.
- (C) Les requêtes ciblées sur des tables de requête qui n'utilisent pas l'autorisation par instance ou par rôle renvoient les mêmes résultats que si l'objet AdminAuthorizationOptions ou AuthorizationOptions était passé à l'API de table de requête. Ce comportement est disponible pour les tables de requête supplémentaires ainsi que pour les tables composites. Aucune vérification des éléments de travail ou des rôles Java EE n'est effectuée ; par conséquent, tous les utilisateurs authentifiés peuvent voir l'intégralité du contenu de la table interrogée. Les clients qui ne veulent pas restreindre la visibilité des objets en appliquant les contraintes d'autorisation par instance ou par rôle fournies par Business Process Choreographer peuvent désactiver les vérifications d'autorisation lors du développement des tables de requête. En revanche, pour les opérations de réclamation et d'achèvement de tâches, les utilisateurs doivent avoir un élément de travail approprié.
- (D) Seule l'autorisation par rôle permet d'accéder aux données de modèle dans les tables de requête prédéfinies ou dans les tables composites configurées avec l'autorisation par rôle. Pour cela, l'appelant utilisant la méthode de requête d'API doit faire partie du rôle Java EE BPESystemAdministrator si l'EJB Business Flow Manager est utilisé, ou du rôle Java EE TaskSystemAdministrator si l'EJB Human Task Manager est utilisé. L'API de table de requête peut être utilisée à la place de l'API de requête pour accéder aux données de modèle.

Eléments de travail et autorisation par instance

Le mécanisme d'autorisation par instance fourni par Business Process Choreographer est basé sur des éléments de travail. Chaque élément de travail décrit qui a quels droits et sur quels objets. Cette information est accessible via la table de requête WORK_ITEM, si l'autorisation par instance est utilisée.

Le tableau suivant décrit les différents types d'éléments de travail pris en considération si l'autorisation par instance est utilisée lors de l'exécution d'une requête sur une table de requête :

Tableau 35. Types d'éléments de travail

Type d'élément de travail	Description
everybody	Permet à tous les utilisateurs d'accéder à un objet spécifique tel qu'une tâche ou une instance de processus. Dans ce cas, l'attribut EVERYBODY de l'élément de travail concerné est mis à TRUE.
individual	Eléments de travail créés pour des utilisateurs particuliers. L'ID d'un utilisateur spécifique est affecté comme valeur à l'attribut OWNER_ID de l'élément de travail concerné. Pour un même objet (tel qu'une tâche), il peut exister plusieurs éléments de travail, chacun avec un attribut OWNER_ID différent.
group	Eléments de travail créés pour les utilisateurs d'un groupe particulier. Le nom d'un groupe spécifique est affecté comme valeur à l'attribut GROUP_NAME de l'élément de travail concerné.
inherited	Les lecteurs et les administrateurs des instances de processus sont également autorisés à hériter de l'accès aux tâches manuelles (y compris celles pour lesquelles il y a escalade) qui appartiennent à ces instances de processus. Dans les requêtes sur les tâches, les vérifications concernant les éléments de travail hérités sont exécutées au moyen de jointures SQL complexes ; ces jointures sont réalisées à l'exécution et ont une incidence sur les performances.

Les éléments de travail sont créés par Business Process Choreographer dans différents cas de figure. Par exemple, à la création d'une tâche, des éléments de travail sont créés pour les différents rôles, tels que ceux de lecteur et de propriétaire potentiel, si des critères d'affectation de personnes ont été spécifiés.

Le tableau suivant décrit les différents types d'éléments de travail créés en fonction des critères d'affectation de personnes définis, si l'autorisation par instance est utilisée lors de l'exécution d'une requête sur une table de requête. Les éléments de travail hérités n'apparaissent pas dans ce tableau, car ils reflètent des relations qui ne peuvent être modélisées explicitement dans la phase de développement.

Tableau 36. Eléments de travail et critères d'affectation de personnes

Type d'élément de travail	Critères d'affectation de personnes associés
everybody	Tous les utilisateurs
individual	Tous les critères d'affectation de personnes excepté les instructions *Personne* (Nobody), *Tous les utilisateurs* (Everybody) et *Groupe* (Group)
group	Groupe

Filtre d'autorisation sur les tables de requête composites

Pour les tables de requête composites, vous pouvez indiquer un filtre d'autorisation si l'autorisation par instance est utilisée. Ce filtre limite les éléments de travail utilisables pour l'autorisation en fonction de certains de leurs attributs. Par exemple, le filtre d'autorisation `"WI.REASON=REASON_ POTENTIAL_OWNER"` appliqué à une table de requête composite dont la table principale est TASK limite à certains types les tâches qui peuvent être renvoyées lorsqu'une personne exécute une requête. Seules sont renvoyées les tâches que cette personne peut avoir à

effectuer ; autrement dit, le résultat est limité aux tâches que la personne est autorisée à réclamer. Ce filtre peut également être indiqué comme filtre de table de requête ou comme filtre de requête, mais il a aussi un effet bénéfique sur les performances des requêtes lorsqu'il est spécifié comme filtre d'autorisation.

Concepts associés

«Tables de requête prédéfinies», à la page 361
Les tables de requête prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requête, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE. Ces tables de requête prédéfinies améliorent les fonctionnalités et les performances des vues de base de données car elles sont optimisées pour exécuter des requêtes portant sur les listes de processus et de tâches.

«Tables de requête supplémentaires», à la page 364
Les tables de requête supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requête un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requête supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requête prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches manuelles.

«tables de requête composites», à la page 366
Dans Business Process Choreographer les tables de requête composites n'ont pas de représentation spécifique de données dans la base de données ; il s'agit d'une combinaison des données de tables de requête prédéfinies et supplémentaires prédéfinies. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

«Options d'autorisation pour l'API de table de requête», à la page 401
Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, vous pouvez passer des options d'autorisation en guise de paramètres d'entrée aux méthodes de l'API de table de requête.

Tâches associées

«Création de tables de requête pour Business Process Choreographer Explorer», à la page 417
Vous pouvez utiliser des tables de requête à la place de l'API query des EJB pour améliorer les performances de Business Process Choreographer Explorer. Pour créer les tables de requête, utilisez Query Table Builder.

Types d'attribut pour les tables de requête

Dans Business Process Choreographer, les types d'attribut sont nécessaires lors de la définition de tables de requête, lors de l'utilisation de valeurs littérales dans les requêtes, ainsi que lors de l'accès aux valeurs dans un résultat de requête. Des règles et des mappages sont disponibles pour chacun des types d'attribut.

Pour définir le type d'un attribut dans une table de requête, on a recours à un sous-ensemble des types caractéristiques du langage de programmation Java et des types propres aux technologies de base de données. Les types d'attribut sont une abstraction des types Java concrets ou des types de données propres aux bases de données. Pour les tables de requête supplémentaires, vous devez utiliser un mappage valide entre types de base de données et types d'attribut.

Le tableau suivant décrit les types d'attribut :

Tableau 37. Types d'attribut

Type d'attribut	Description
ID	L'ID servant à identifier une tâche manuelle (TKIID), une instance de processus (PIID) ou d'autres objets. Par exemple, pour réclamer ou effectuer une tâche manuelle particulière, l'utilisateur doit spécifier celle-ci en l'identifiant par son attribut TKIID.
STRING	Les descriptions des tâches ou les propriétés des requêtes peuvent être représentées par des chaînes.
NUMBER	Les nombres sont utilisés pour les attributs tels que le niveau de priorité d'une tâche.
TIMESTAMP	Les horodatages décrivent un instant précis ; par exemple, la date et l'heure de création d'une tâche manuelle ou la date et l'heure de fin d'exécution d'une instance de processus.
DECIMAL	Le type DECIMAL peut être utilisé pour les propriétés d'une requête ; par exemple, pour définir une propriété de requête avec une variable du type XSD 'double'.
BOOLEAN	Les booléens peuvent prendre une valeur parmi deux : true ou false. Par exemple, les tâches manuelles ont un attribut, autoClaim, qui précise si oui ou non une tâche est réclamée automatiquement lorsqu'il n'existe qu'un seul utilisateur qui puisse être son propriétaire.

Concepts associés

«Correspondance entre types des bases de données et types d'attribut»
Utilisez des types d'attribut pour définir des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

«Correspondance entre types d'attribut et représentations littérales», à la page 390
Utilisez des types d'attribut pour définir des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête. Cette rubrique décrit la correspondance entre types d'attribut et représentations littérales.

«Correspondance entre types d'attribut et paramètres», à la page 392
Utilisez des types d'attribut lorsque vous définissez des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

«Correspondance entre types d'attribut et types d'objet Java», à la page 393
Utilisez des types d'attribut pour définir des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête. Cette rubrique décrit la correspondance entre types d'attribut et types d'objet Java.

«Compatibilité entre types d'attribut», à la page 394
Utilisez des types d'attribut lorsque vous définissez des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Correspondance entre types des bases de données et types d'attribut :

Utilisez des types d'attribut pour définir des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Le tableau suivant décrit les types propres aux bases de données et indique leur correspondance avec les types d'attribut :

Tableau 38. Correspondance entre types des bases de données et types d'attribut

Type de base de données	Type d'attribut
Type binaire sur 16 octets. Il s'agit du type utilisé pour les ID tels que le TKIID des tâches dans les tables de Business Process Choreographer.	ID
Type caractère. Sa longueur dépend de la colonne de table de base de données qui est référencée par l'attribut dans la table de requête.	STRING
Type d'entier de base de données, tel qu'un 'integer', un 'short' ou un 'long'.	NUMBER
Type d'horodatage de base de données.	TIMESTAMP
Type décimal, tel qu'un 'float' ou un 'double'.	DECIMAL
Type convertible en valeur booléenne telle qu'un nombre. 1 est interprété comme *true* (vrai) et tous les autres nombres, comme *false* (faux).	BOOLEAN

Généralement, les tables de requête supplémentaires font référence à des tables et des vues existantes de la base de données ; lorsque c'est le cas, elles ne nécessitent pas la création de tables ou de vues spécifiques.

Exemple

Prenons le cas d'un environnement DB2. Une table appelée CUSTOM.ADDITIONAL_INFO doit être représentée en tant que table de requête supplémentaire dans Business Process Choreographer. L'instruction SQL suivante crée la table de base de données :

```
CREATE TABLE CUSTOM.ADDITIONAL_INFO
(
  PIID       CHAR(16) FOR BIT DATA,
  INFO       VARCHAR(220),
  COUNT      INTEGER
);
```

La correspondance suivante, entre les types des colonnes côté base de données et les types d'attribut côté table de requête, est utilisé pour définir une table de requête supplémentaire chargée de représenter la table CUSTOM.ADDITIONAL_INFO.

Tableau 39. Exemple de mappage entre types de base de données et types d'attribut

Colonne et type de base de données	Attribut et type de table de requête
PIID CHAR(16) FOR BIT DATA	PIID (ID)
INFO VARCHAR(220)	INFO (STRING)
COUNT INTEGER	COUNT (NUMBER)

Correspondance entre types d'attribut et représentations littérales :

Utilisez des types d'attribut pour définir des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête. Cette rubrique décrit la correspondance entre types d'attribut et représentations littérales.

Des valeurs littérales peuvent être utilisées dans les expressions pour définir des critères de filtrage et de sélection, comme dans les filtres des tables de requête composites ainsi que dans les filtres passés à l'API de table de requête.

Le tableau suivant décrit les types d'attribut et indique leur correspondance avec les valeurs littérales. Les marques de réservation figurent en *italique*. Notez que les types d'attribut ID et TIMESTAMP, qui peuvent être passés à l'API de table de requête, utilisent une syntaxe spéciale, également employée par l'API de requête.

Tableau 40. Correspondance entre types d'attribut et valeurs littérales

Type d'attribut	**Syntaxe et utilisation comme valeur littérale dans les expressions**
ID	`ID ('représentation chaîne d'un ID' )`
	Lors du développement d'applications clientes, les ID sont représentés soit comme des chaînes, soit comme des instances de l'interface com.ibm.bpe.api.OID. Il est possible d'obtenir la représentation sous forme de chaîne d'une instance de l'interface com.ibm.bpe.api.OID en utilisant sa méthode toString. La chaîne doit être encadrée d'apostrophes.
STRING	`'la chaîne'`
	La chaîne doit être encadrée d'apostrophes.
NUMBER	`nombre`
	Le nombre sous forme de texte, non délimité par des apostrophes. Des constantes sont définies et utilisables pour certains attributs du type NUMBER dans les tables de requête prédéfinies.
TIMESTAMP	`TS ('AAAA-MM-JJThh:mm:ss')`
	L'horodatage doit être spécifié au format suivant : • `AAAA` : les quatre chiffres de l'année • `MM` : les deux chiffres du mois de l'année • `DD` : les deux chiffres du jour du mois • `hh` : les deux chiffres de l'heure (sur 24 heures) • `mm` : les deux chiffres de la minute • `ss` : les deux chiffres de la seconde. L'interprétation de l'horodatage se fait par rapport au fuseau horaire de l'utilisateur.
DECIMAL	`nombre.fraction`
	Le nombre décimal sous forme de texte, non délimité par des apostrophes ; la partie `.fraction` est optionnelle.
BOOLEAN	`true`, `false`
	La valeur booléenne sous forme de texte.

Exemple

- `filterOptions.setQueryCondition("STATE=2");`
- `filterOptions.setQueryCondition("STATE=STATE_READY");`

- un critère de sélection sur une table de requête attachée TASK_DESC : "LOCALE='en_US'"
- filterOptions.setQueryCondition("PTID=ID('_PT:8001011e.1dee8e51.247d6df6.29a60000')");

Correspondance entre types d'attribut et paramètres :

Utilisez des types d'attribut lorsque vous définissez des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Le tableau suivant décrit les types d'attribut et leur correspondance avec les valeurs de paramètre qui peuvent être utilisées dans les expressions pour définir des critères de filtrage et de sélection, comme dans les filtres des tables de requête composites ainsi que dans les filtres passés à l'API de table de requête.

Tableau 41. Correspondance entre types d'attribut et valeurs de paramètre utilisateur

Type d'attribut	**Utilisation comme valeur de paramètre dans les expressions**
ID	PARAM*(nom)* Lors du développement d'applications clientes, les ID sont représentés soit comme des chaînes, soit comme des instances de l'interface com.ibm.bpe.api.OID. Les deux représentations sont valides et acceptées comme paramètres. Un tableau d'octets (byte[]) reflétant un OID valide peut également être utilisé.
STRING	PARAM*(nom)* La représentation chaîne de l'objet qui, à l'exécution, est passée à l'API de table de requête par la méthode toString.
NUMBER	PARAM*(nom)* Une représentation du nombre sous forme de java.lang.Long, java.lang.Integer, java.lang.Short ou java.lang.String est passée à l'API de table de requête. Les noms des constantes définies pour certains attributs des tables de requête prédéfinies peuvent aussi être passés.
TIMESTAMP	PARAM*(nom)* Les représentations suivantes sont valides : • Une représentation java.lang.String de l'horodatage • Instances de com.ibm.bpe.api.UTCDate • Instances de java.util.Calendar
DECIMAL	PARAM*(nom)* Une représentation du nombre décimal sous forme de java.lang.Long, java.lang.Integer, java.lang.Short, java.lang.Double, java.lang.Float ou java.lang.String est passée à l'API de table de requête.

Tableau 41. Correspondance entre types d'attribut et valeurs de paramètre utilisateur (suite)

Type d'attribut	Utilisation comme valeur de paramètre dans les expressions
BOOLEAN	PARAM*(nom)* Les valeurs admises sont : • Une représentation java.lang.String du booléen • Un java.lang.Short, java.lang.Integer ou java.lang.Long avec des valeurs appropriées ; 0 (pour false) ou 1 (pour true) • Un objet java.lang.Boolean

Exemple

```
...
// l'exemple suivant illustre l'exécution d'une requête sur une table de
// requêtes composite, COMP.TASKS, avec un paramètre "client"
java.util.List params = new java.util.ArrayList();

list.add(new com.ibm.bpe.api.Parameter("client", "IBM");
// l'EJB (Enterprise JavaBeans) de Business Flow Manager ou l'EJB
// de Human Task Manager Enterprise permet d'accéder aux tables de requête
service.bfm.queryEntities("COMP.TASKS", null, null, params);
...
```

Correspondance entre types d'attribut et types d'objet Java :

Utilisez des types d'attribut pour définir des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête. Cette rubrique décrit la correspondance entre types d'attribut et types d'objet Java.

Le tableau suivant décrit les types d'attribut et indique leur correspondance avec les types d'objet Java dans les ensembles de résultats des requêtes.

Tableau 42. Correspondance entre types d'attribut et types d'objet Java

Type d'attribut	Type d'objet Java associé
ID	com.ibm.bpe.api.OID
STRING	java.lang.String
NUMBER	java.lang.Long
TIMESTAMP	java.util.Calendar
DECIMAL	java.lang.Double
BOOLEAN	java.lang.Boolean

Exemple

```
...
// l'exemple suivant illustre l'exécution d'une requête sur une table de
// requêtes composite appelée COMP.TA ; l'attribut "STATE" est du type d'attribut NUMBER
...
// exécution de la requête
// l'EJB (Enterprise JavaBeans) de Business Flow Manager ou l'EJB
// de Human Task Manager Enterprise permet d'accéder aux tables de requête
EntityResultSet rs = bfm.queryEntities("COMP.TA",null,null,params);

// obtenir les entités et itérer dessus
List entities = rs.getEntities();
```

```
for (int i = 0 ; i < entities.size(); i++) {

      // manipuler une entité particulière
      Entity en = (Entity) entities.get(i);

      // notez que le code suivant pourrait être écrit d'une
      // façon moins spécifique en utilisant les objets d'informations
      // d'attributs renvoyés par la méthode ei.getAttributeInfo()

      // obtenir l'attribut STATE
      Long state = (Long) en.getAttributeValue("STATE");
      ...
}
...
```

Compatibilité entre types d'attribut :

Utilisez des types d'attribut lorsque vous définissez des tables de requête dans Business Process Choreographer et lorsque vous lancez des requêtes sur ces tables, ainsi que pour accéder aux valeurs d'un résultat de requête.

Le tableau suivant dresse la liste des types d'attribut et indique leur compatibilité. Utilisez ces informations pour définir des filtres et des critères de sélection dans les tables de requête. La compatibilité entre types d'attribut est signalée par un **X**.

Tableau 43. Compatibilité entre types d'attribut

Type d'attribut	ID	STRING	NUMBER	TIMESTAMP	DECIMAL	BOOLEAN
ID	X					
STRING		X				
NUMBER			X		X	
TIMESTAMP				X		
DECIMAL			X		X	
BOOLEAN						X

Dans les expressions de table de requête qui spécifient des filtres et des critères de condition, les types des attributs ou des valeurs comparés doivent être compatibles. Par exemple, `WI.OWNER_ID=1` est un filtre non valide, car l'opérande côté gauche est du type STRING alors que l'opérande côté droit est du type NUMBER.

Requêtes sur des tables de requête

L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête, qui est disponible pour l'interface EJB de Business Flow Manager et l'API REST.

Une requête s'exécute toujours sur une seule table de requête. Le contenu des tables de requête est extrait au moyen des méthodes de l'API qui, pour certaines, sont basées sur les entités et pour d'autres, sur les lignes. Des paramètres (ou arguments) d'entrée sont passés aux méthodes de l'API de table de requête.

Concepts associés

«Méthodes de l'API de table de requête»
L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête. Le contenu des tables de requête est extrait au moyen des méthodes de l'API qui, pour certaines, sont basées sur les entités et pour d'autres, sur les lignes.

«Paramètres de l'API de table de requête», à la page 397
Les méthodes de l'API de table de requête permettent d'extraire du contenu lors de l'exécution de requêtes sur une table de requête dans Business Process Choreographer.

«Résultats des requêtes exécutées sur les tables de requête», à la page 404
Les méthodes de l'API de table de requête s'utilisent lors de l'exécution de requêtes sur une table de requête dans Business Process Choreographer. Le résultat de la méthode queryEntityCount ou de la méthode queryRowCount est un simple nombre. En revanche, les méthodes queryEntities et queryRows renvoient des ensembles de résultats.

«Développement des tables de requête», à la page 374
Dans Business Process Choreographer, le développement des tables de requête supplémentaires et des tables de requête composites s'effectue au cours du développement de l'application, à l'aide de l'outil Query Table Builder. Les tables de requête prédéfinies ne peuvent pas être développées ni déployées. Disponibles lorsque Business Process Choreographer est installé, elles fournissent une vue simplifiée des artefacts du schéma de base de données Business Process Choreographer.

«Tables de requête prédéfinies», à la page 361
Les tables de requête prédéfinies fournissent l'accès aux données dans la base de données de Business Process Choreographer. Elles correspondent à une représentation, sous forme de tables de requête, des vues de la base de données de Business Process Choreographer, telles que TASK ou PROCESS_INSTANCE. Ces tables de requête prédéfinies améliorent les fonctionnalités et les performances des vues de base de données car elles sont optimisées pour exécuter des requêtes portant sur les listes de processus et de tâches.

«Tables de requête supplémentaires», à la page 364
Les tables de requête supplémentaires que vous définissez dans Business Process Choreographer fournissent à l'API de table de requête un accès à des données métier dites "externes", c'est-à-dire qui ne sont pas gérées par Business Process Choreographer. Grâce à ces tables de requête supplémentaires, les données externes correspondantes peuvent être utilisées en association avec les données des tables de requête prédéfinies pour extraire des informations sur des instances de processus métier ou sur des tâches manuelles.

«tables de requête composites», à la page 366
Dans Business Process Choreographer les tables de requête composites n'ont pas de représentation spécifique de données dans la base de données ; il s'agit d'une combinaison des données de tables de requête prédéfinies et supplémentaires prédéfinies. Utilisez-les pour obtenir les informations d'une liste d'instances de processus ou d'une liste de tâches (liste Mes tâches, par exemple).

«Filtres et critères de sélection des tables de requête», à la page 378
Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requête, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requête.

Méthodes de l'API de table de requête :

L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête. Le contenu des tables de requête est extrait au moyen des méthodes de l'API qui, pour certaines, sont basées sur les entités et pour d'autres, sur les lignes.

Les méthodes suivantes sont disponibles dans l'API de table de requête pour permettre l'exécution de requêtes sur les tables de requête dans Business Process Choreographer :

Tableau 44. Méthodes pour les requêtes exécutées sur les tables de requête

Fonction	Méthodes
Interrogation de contenu	• queryEntities • queryRows Les deux méthodes renvoient du contenu de la table de requête. La méthode queryEntities renvoie un contenu basé sur des entités et la méthode queryRows, un contenu basé sur des lignes.
Interrogation du nombre d'objets	• queryEntityCount • queryRowCount Les deux méthodes renvoient le nombre d'objets présents dans la table de requête, mais ce nombre peut varier en fonction de l'approche adoptée (requête par entité ou par ligne).

Les requêtes par entité, exécutées à l'aide des méthodes queryEntities et queryEntityCount, supposent qu'une table de requête contient des entités identifiables de façon unique, tel que défini par la clé primaire dans la table de requête principale.

Les requêtes par ligne, exécutées à l'aide des méthodes queryRows et queryRowCount, renvoient un ensemble de résultats comme avec JDBC ; cet ensemble de résultats est également à base de lignes et navigable au moyen des méthodes first et next. L'ensemble de résultats obtenu lorsque vous exécutez une requête sur une table de requête en utilisant l'API de table de requête est comparable aux ensembles QueryResultSet renvoyés par l'API de requêtes. En général, le nombre de lignes est supérieur au nombre d'entités contenues dans une table de requête. Une même entité, par exemple une tâche manuelle identifiée par son ID de tâche TKIID, peut apparaître plusieurs fois dans l'ensemble de résultats de lignes.

Une instance particulière contenue dans une table de requête prédéfinie n'existe qu'en un seul exemplaire dans un environnement Business Process Choreographer. Les tâches manuelles et les processus métier sont des exemples de telles instances. Chaque instance est identifiée de manière unique par un ID ou un ensemble d'ID. Il s'agit du TKIID pour les instances de tâche manuelle et du PIID pour les instances de processus.

Les tables de requête composites comprennent une table de requête principale et zéro, une ou plusieurs tables de requête attachées. L'identification des objets dans une table composite se fait par les ID unique des objets de la table de requête principale. C'est donc la table de requête principale qui détermine le type d'entité d'une table composite. Par exemple, une table de requête composite dont la table principale est TASK contiendra obligatoirement des entités du type TASK. La

relation un à un ou un à zéro qui existe entre la table principale et les tables attachées garantit que les tables attachées n'introduisent pas d'entités en double.

Les requêtes par entité tirent parti du fait qu'une table de requête contient des entités identifiables de façon unique, tel que défini par la clé primaire dans la table de requête principale. La garantie d'unicité des instances et d'absence de doublons est une qualité particulièrement appréciée par les développeurs d'applications clientes, notamment ceux qui sont chargés de la partie interface utilisateur. Par exemple, il est essentiel qu'une tâche manuelle ne soit affichée qu'une seule fois dans l'interface utilisateur. Des instances uniques sont renvoyées si l'API de table de requête par entité est utilisée.

Les requêtes par ligne peuvent renvoyer des doubles des lignes de la table de requête principale si l'autorisation par instance est utilisée.

- Les informations de la table de requête WORK_ITEM sont récupérées avec la requête. Par exemple, si l'attribut `WI.REASON` est extrait en plus des attributs définis dans la table de requête, plusieurs lignes sont susceptibles d'être renvoyées comme résultat. En effet, cet attribut stocke le motif d'accès à une entité telle qu'une tâche ou une instance de processus ; or, un utilisateur peut accéder à une telle entité pour plusieurs raisons.
- L'autorisation par instance est utilisée et l'opérateur 'distinct' n'est pas spécifié. Bien qu'il n'y ait pas d'extraction des données d'élément de travail, plusieurs lignes peuvent être renvoyées si l'autorisation par instance est utilisée.

Si l'API de table de requête par entité est utilisée :

- Les requêtes par entité sont toujours exécutées avec l'opérateur SQL 'distinct'.
- Les requêtes par entité renvoient des résultats dans lesquels les informations relatives aux éléments de travail peuvent être fournies sous forme de valeurs de tableau (array).

Paramètres de l'API de table de requête :

Les méthodes de l'API de table de requête permettent d'extraire du contenu lors de l'exécution de requêtes sur une table de requête dans Business Process Choreographer.

Les paramètres d'entrée suivants sont passés aux méthodes de l'API de table de requête :

Tableau 45. Paramètres de l'API de table de requête

Paramètre	Optionnel	Type et description
Nom de la table de requête	Non	java.lang.String
		Nom unique de la table de requête.
Options de filtrage	Oui	com.ibm.bpe.api.FilterOptions si l'EJB Flow Manager Enterprise est utilisé ou com.ibm.task.api.FilterOptions si l'EJB Human Task Manager est utilisé.
		Options utilisables pour définir la requête. Par exemple, un seuil peut être fixé dans ce paramètre pour limiter le nombre de résultats renvoyés.

Tableau 45. Paramètres de l'API de table de requête (suite)

Paramètre	Optionnel	Type et description
Options d'autorisation	Oui	com.ibm.bpe.api.AuthorizationOptions ou com.ibm.bpe.api.AdminAuthorizationOptions si l'EJB Business Flow Manager est utilisé. com.ibm.task.api.AuthorizationOptions ou com.ibm.task.api.AdminAuthorizationOptions si l'EJB Human Task Manager est utilisé.
		Si l'autorisation par instance est utilisée, ce paramètre permet d'imposer des contraintes encore plus serrées. Pour les tables de requête nécessitant une autorisation par rôle, une instance d'objet AdminAuthorizationOptions doit être passée.
Paramètres	Oui	Un élément java.util.List de com.ibm.bpe.api.Parameter si l'EJB Business Flow Manager est utilisé, ou l'élément com.ibm.task.api.Parameter si l'EJB Human Task Manager est utilisé.
		Ce paramètre sert à passer des paramètres utilisateur qui sont spécifiés dans un critère de filtrage ou de sélection sur une table de requête composite.

Une requête s'exécute toujours sur une seule table de requête. La relation entre plusieurs tables de requête est définie au moyen d'une table de requête composite. Ce concept particulier à l'API de table de requête correspond à la notion de vues de base de données dans l'API de requête.

Vous indiquez des filtres et des critères de sélection dans les expressions lors du développement des tables de requête à l'aide de Query Table Builder. Pour plus d'informations, voir les rubriques du centre de documentation relatives aux tables de requête composites et aux critères de filtre et de recherche des tables de requête. Pour plus d'information sur Query Table Builder, voir le site des SupportPacs de WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

Concepts associés

«Nom de la table de requête»
Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, le nom de cette table est passé comme paramètre d'entrée aux méthodes de l'API de table de requête.

«Options de filtrage des tables de requête»
Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, vous pouvez passer des options de filtrage en guise de paramètres d'entrée aux méthodes de l'API de table de requête.

«Options d'autorisation pour l'API de table de requête», à la page 401
Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, vous pouvez passer des options d'autorisation en guise de paramètres d'entrée aux méthodes de l'API de table de requête.

«Paramètres», à la page 403
Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, vous pouvez passer des paramètres utilisateur aux méthodes de l'API de table de requête. Dans une définition de table de requête, vous pouvez spécifier des paramètres dans les filtres appliqués à la table de requête principale, aux autorisations et à la table de requête elle-même. Des paramètres peuvent aussi être spécifiés dans les critères de sélection sur les tables de requête attachées.

Nom de la table de requête :

Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, le nom de cette table est passé comme paramètre d'entrée aux méthodes de l'API de table de requête.

Le nom de la table de requête dont il est question ici est celui de la table constituant la cible de l'exécution de la requête.

- Lorsque la cible est une table de requête prédéfinie, le nom spécifié est donc celui de la table prédéfinie en question.
- Dans le cas d'une table composite ou supplémentaire, il s'agit du nom de la table de requête qui a été spécifié lors de la modélisation de la table. Le nom d'une table composite ou supplémentaire est de la forme *préfixe.nom* ; la partie *préfixe* ne doit pas être `'IBM'`.

Le nom de la table de requête et le préfixe doivent tous les deux être en lettres majuscules. Le nom de la table de requête ne doit pas dépasser 28 caractères.

Options de filtrage des tables de requête :

Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, vous pouvez passer des options de filtrage en guise de paramètres d'entrée aux méthodes de l'API de table de requête.

Une instance de la classe com.ibm.bpe.api.FilterOptions si l'EJB Business Flow Manager Enterprise est utilisé, ou une instance de la classe com.ibm.task.api.FilterOptions si l'EJB Human Task Manager est utilisé, peut être transmise à l'API de table de requête. Les options de filtrage permettent de configurer une requête en agissant sur les caractéristiques suivantes :

- Un seuil et un décalage (skipCount)
- Des attributs de tri (similaires à la clause ORDER BY dans une requête SQL)
- Filtre de requête fourni par l'utilisateur

- L'ensemble d'attributs renvoyé, y compris les données d'élément de travail
- Autre

L'ensemble de résultats qui peut être obtenu d'une table de requête est spécifié par la définition de cette table. Cependant, vous pouvez avoir besoin de spécifier des options supplémentaires au moment d'exécuter la requête. Le tableau suivant décrit les options de filtrage qui peuvent être spécifiées au moyen de l'objet FilterOptions.

Tableau 46. Paramètres de l'API de table de requête : options de filtrage

Option	Type	Description
Attributs sélectionnés	java.lang.String	• Une liste (séparée par des virgules) des attributs de la table de requête à renvoyer dans l'ensemble de résultats. • Si l'autorisation par instance est utilisée, vous pouvez extraire les données d'élément de travail en spécifiant les attributs de la table de requête WORK_ITEM, préfixés avec `WI.` (par exemple, `WI.REASON`). • Si 'null' est spécifié, tous les attributs de la table de requête sont renvoyés, sans les données d'élément de travail.
Attributs de tri	java.lang.String	Une liste (séparée par des virgules) des attributs de la table de requête éventuellement suivis de ASC ou DESC pour spécifier un tri par ordre croissant ou décroissant, respectivement. Cette liste est similaire à la clause SQL ORDER BY : *sortAttributes* ::= *attribut* [ASC|DESC] [, *sortAttributes*]. Si ASC ou DESC n'est pas spécifié, l'ordre de tri croissant (ASC) est appliqué par défaut. L'ordre dans lequel les attributs de tri sont spécifiés a son importance. Dans cet exemple, les tâches de la table de requête TASK sont triées par état dans un ordre décroissant, et par ordre croissant dans les groupes ayant le même état STATE by NAME : `"STATE DESC, NAME ASC".`
Seuil	java.lang.Integer	Définit une limite qui peut être : • Le nombre maximum de lignes renvoyées si queryRows est utilisée. • Le nombre maximum d'entités renvoyées si queryEntities est utilisée. Le nombre réel d'entités disponibles dans la table de requête concernée peut dépasser la limite définie pour la requête, même si l'ensemble de résultats ne contient pas autant d'entités que le nombre correspondant à cette limite. Cela est dû à des impératifs techniques qui entrent en jeu lorsque les données d'élément de travail sont sélectionnées. • Le compte renvoyé si queryRowCount ou queryEntityCount est utilisée. La valeur par défaut est 'null', ce qui signifie qu'aucun seuil n'est défini.
Nombre d'éléments à sauter (skipCount)	java.lang.Integer	Définit le nombre de lignes (requêtes par ligne) ou le nombre d'entités (requêtes par entité) à sauter. Comme pour le paramètre de seuil, skipCount peut manquer d'exactitude dans le cas des requêtes par entité. Le rôle du paramètre skipCount est de permettre la pagination lorsque de gros ensembles de résultats sont renvoyés. La valeur par défaut est 'null', ce qui signifie que skipCount n'est pas défini.

Tableau 46. Paramètres de l'API de table de requête : options de filtrage (suite)

Option	Type	Description
Fuseau horaire	java.util.TimeZone	Le fuseau horaire utilisé lors de la conversion des horodatages. L'attribut CREATED de la table requêtes prédéfinie TASK est un exemple d'horodatage affecté par ce paramètre. S'il n'est pas spécifié (null), le fuseau horaire utilisé est celui du serveur.
Environnement local	java.util.Locale	L'environnement local utilisé pour calculer la valeur du paramètre système `$LOCALE`. Exemple d'utilisation de `$LOCALE` dans un critère de sélection : `'LOCALE=$LOCALE'`.
Lignes distinctes	java.lang.Boolean	S'applique uniquement aux requêtes par ligne. Si la valeur est true, les requêtes par ligne renvoient des lignes distinctes. Cela n'implique pas que des lignes uniques sont renvoyées en raison de l'éventuelle multiplicité des données d'élément de travail.
Condition de requête	java.lang.String	Cette option effectue un filtrage supplémentaire sur l'ensemble de résultats. Tous les attributs définis pour la table de requête peuvent être référencés. Si une autorisation est requise pour la table de requête, les colonnes définies pour la table de requête WORK_ITEM peuvent également être référencées à l'aide du préfixe `WI` (par exemple, `WI.REASON=REASON_POTENTIAL_OWNER`).

Options d'autorisation pour l'API de table de requête :

Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, vous pouvez passer des options d'autorisation en guise de paramètres d'entrée aux méthodes de l'API de table de requête.

Utilisez une instance de la classe com.ibm.bpe.api.AuthorizationOptions ou de la classe com.ibm.bpe.api.AdminAuthorizationOptions si l'EJB Business Flow Manager est utilisé, ou une instance de la classe com.ibm.task.api.AuthorizationOptions ou de la classe com.ibm.task.api.AdminAuthorizationOptions si l'EJB Human Task Manager est utilisé, pour indiquer des options d'autorisation lorsque la requête est exécutée.

Si une autorisation par instance est utilisée, les instances de la classe AuthorizationOptions permettent de spécifier les types d'éléments de travail identifiant les instances qui peuvent être renvoyées par la requête.

Une instance de la classe AuthorizationOptions peut être transmise à l'API de table de requête si la requête est exécutée sur une table de requête prédéfinie contenant des données d'instance. Elle peut aussi être passée à l'API si la requête est exécutée sur une table composite dont la table principale contient des données d'instance et si l'autorisation par instance est configurée pour être utilisée. Si la requête est exécutée sur une table prédéfinie avec des données de modèle ou sur une table composite avec l'autorisation par rôle configurée, une exception `com.ibm.bpe.api.EngineNotAuthorizedException` est émise si l'EJB Business Flow Manager est utilisé, ou `com.ibm.task.api.NotAuthorizedException` si l'EJB Human Task Manager est utilisé. Dans tous les autres cas, les options d'autorisation passées à l'API de table de requête sont ignorées.

Les tables de requête composites peuvent restreindre les types d'éléments de travail pris en compte lors de l'identification des objets (ou entités) qu'elles contiennent. Par exemple, si les options d'autorisation passées à l'API de table de requête sont configurées pour utiliser les éléments de travail 'everybody', elles ne sont prises en compte que si l'utilisation de tels éléments de travail est prévue dans

la définition de la table de requête composite. Voici une règle simple à retenir : l'API de table de requête ne peut pas forcer la prise en compte d'un type d'élément de travail dont l'utilisation n'est pas spécifiée dans la définition de la table de requête ; en revanche, l'API n'est pas forcée de prendre en compte un type d'élément de travail dont l'utilisation est spécifiée dans la définition de la table de requête. De même, l'API de table de requête ne peut pas spécifier un type d'autorisation différent de celui qui est défini pour une table de requête composite ou prédéfinie.

En fonction du type de la table de requête interrogée, différentes options d'autorisation par défaut s'appliquent si l'objet d'autorisation n'est pas spécifié ou si, comme c'est le cas par défaut, les attributs associés (everybody, individual, group ou inherited) sont mis à 'null'.

Le tableau suivant indique les options d'autorisation appliquées par défaut à l'autorisation par instance, pour chaque combinaison de type de table de requête et de type d'élément de travail.

Tableau 47. Paramètres de l'API de table de requête : options d'autorisation par défaut pour l'autorisation par instance

Type de table de requête	**Elément de travail 'everybody'**	**Elément de travail 'individual'**	**Elément de travail 'group'**	**Elément de travail 'inherited'**
Prédéfinie avec des données d'instance	TRUE	TRUE	TRUE	FALSE
Prédéfinie avec des données de modèle	N/A	N/A	N/A	N/A
Composite avec une table principale contenant des données d'instance	TRUE	TRUE	TRUE	TRUE
Composite avec une table principale contenant des données de modèle	N/A	N/A	N/A	N/A
Supplémentaire	N/A	N/A	N/A	N/A

N/A (non applicable) signifie que l'autorisation par instance n'est pas utilisée et que, par conséquent, toute option concernant les éléments de travail dans l'objet d'autorisation est ignorée.

Si TRUE est spécifié, la requête résultante ne prend en compte le type d'élément de travail concerné que si la table de requête est définie pour utiliser ce type d'élément de travail. Cette règle est valable pour toutes les tables de requête prédéfinies qui contiennent des données d'instance ; elle peut ne pas l'être pour une table de requête composite. En ce qui concerne l'élément de travail 'group', celui-ci doit être activé sur le conteneur de tâches manuelles. Un exemple de cas

où l'élément de travail est réglé à TRUE est celui où l'administrateur d'une instance de processus peut voir les instances de tâches manuelles participantes qui sont créées pour cette instance de processus.

Indiquez une instance de la classe AdminAuthorizationOptions plutôt qu'une instance de la classe AuthorizationOptions si :

- Une requête est exécutée sur une table de requête configurée avec l'autorisation par rôle. Les tables de requête prédéfinies avec des données de modèle requièrent l'autorisation par rôle ; de même, les tables composites dont la table principale contient des données de modèle peuvent être configurées pour exiger l'autorisation par rôle.
- Une requête est exécutée sur une table de requête contenant des données d'instance ou sur une table composite dont la table principale contient des données d'instance. Le contenu de cette table de requête doit être renvoyé, sans considération des restrictions liées à l'autorisation d'un utilisateur particulier. Ce comportement particulier à l'API de table de requête correspond à l'utilisation de la méthode queryAll dans l'API de requête.
- Une requête doit être exécutée pour le compte d'un autre utilisateur.

Le tableau suivant décrit de quelle manière les comportements décrits plus haut sont obtenus :

Tableau 48. Paramètres de l'API de table de requête : AdminAuthorizationOptions

Situation	Description
Valeur de onBehalfUser = null	• Si une requête est exécutée sur une table de requête configurée avec l'autorisation par rôle, l'intégralité de son contenu est renvoyée. • Si la requête est exécutée sur une table de requête qui utilise l'autorisation par instance, les objets qu'elle contient ne sont pas vérifiés par rapport aux éléments de travail autorisés pour tel ou tel utilisateur. La requête renvoie donc tous les objets contenus dans la table de requête.
Valeur de onBehalfUser = un utilisateur particulier	La requête est exécutée avec l'autorité de l'utilisateur spécifié, et les objets contenus dans la table de requête cible sont vérifiés par rapport aux éléments de travail autorisés pour cet utilisateur, si l'autorisation par instance est utilisée.

Si vous indiquez AdminAuthorizationOptions, l'appelant doit faire partie du rôle Java EE BPESystemAdministrator ou BPESystemMonitor si l'EJB Business Flow Manager est utilisé, ou du rôle Java EE TaskSystemAdministrator ou TaskSystemMonitor si l'EJB Human Task Manager est utilisé.

Concepts associés

«Autorisation pour les tables de requête», à la page 383
Vous pouvez utiliser une autorisation par instance, une autorisation par rôle ou aucune autorisation lorsque vous exécutez des requêtes ou des tables de requête.

Paramètres :

Lorsque vous exécutez une requête sur une table de requête dans Business Process Choreographer, vous pouvez passer des paramètres utilisateur aux méthodes de l'API de table de requête. Dans une définition de table de requête, vous pouvez spécifier des paramètres dans les filtres appliqués à la table de requête principale,

aux autorisations et à la table de requête elle-même. Des paramètres peuvent aussi être spécifiés dans les critères de sélection sur les tables de requête attachées.

Les paramètres système $USER et $LOCALE sont remplacés à l'exécution dans les filtres et les critères de sélection et n'ont pas besoin d'être passés à l'API de table de requête. La valeur d'entrée utilisée pour le calcul du paramètre système $LOCALE est fournie en définissant l'environnement local dans les options de filtre.

Les paramètres utilisateur doivent être passés à l'API de table de requête au moment où la requête est exécutée. Il convient pour cela de transmettre une liste d'instances de la classe com.ibm.bpe.api.Parameter si l'EJB Business Flow Manager est utilisé, ou une instance de la classe com.ibm.task.api.Parameter si l'EJB Human Task Manager est utilisé.

Les propriétés suivantes doivent être spécifiées dans un objet Parameter :

Tableau 49. Paramètres utilisateur destinés à l'API de table de requête

Propriété	Description
Nom	Le nom du paramètre tel qu'il est spécifié dans la définition de table de requête. Le nom est sensible à la casse.
Valeur	La valeur du paramètre. Le type du paramètre doit être compatible avec le type de l'opérande côté gauche et de tous les filtres et critères de sélection où ce paramètre est utilisé. Les constantes définies pour certains attributs des tables de requête prédéfinies peuvent être passés comme des chaînes ; par exemple, STATE_READY.

Exemple

```
// exécute une requête sur une table de requête composite
// CUST.CPM avec le filtre de table de requête principale
// 'STATE=PARAM(theState)'
EntityResultSet ers = null;
List parameterList = new ArrayList();
parameterList.add(new Parameter
("theState", new Integer(2)));

// exécution de la requête ;
// l'EJB Business Flow Manager EJB ou l'EJB Human Task
// Manager permet d'accéder aux tables de requête
ers = bfm.queryEntities
("CUST.CPM", null, null, parameterList);

// manipuler l'ensemble de résultats
// ...
```

Résultats des requêtes exécutées sur les tables de requête :

Les méthodes de l'API de table de requête s'utilisent lors de l'exécution de requêtes sur une table de requête dans Business Process Choreographer. Le résultat de la méthode queryEntityCount ou de la méthode queryRowCount est un simple nombre. En revanche, les méthodes queryEntities et queryRows renvoient des ensembles de résultats.

EntityResultSet

Une instance de la classe com.ibm.bpe.api.EntityResultSet est renvoyée par la méthode queryEntities si l'EJB Business Flow Manager est utilisé. Une instance de

la classe com.ibm.task.api.EntityResultSet est renvoyée par la méthode queryEntities si l'EJB Human Task Manager est utilisé. Un ensemble de résultats d'entités a les propriétés suivantes :

Tableau 50. Propriétés d'un ensemble de résultats d'entités renvoyé par l'API de table de requête

Propriété	Description
queryTableName	Nom de la table de requête sur laquelle la requête a été exécutée.
entityTypeName	• Si la requête a été exécutée sur une table de requête composite, cette propriété est le nom de la table de requête principale. • Si la requête a été exécutée sur une table de requête prédéfinie ou supplémentaire, cette propriété est le nom de la table en question ; autrement dit, elle a la même valeur que la propriété *queryTableName*.
EntityInfo	Cette propriété contient les méta-informations des entités contenues dans l'ensemble de résultats. Une liste java.util.List d'objets com.ibm.bpe.api.AttributeInfo si l'EJB Business Flow Manager est utilisé, ou une liste d'objets com.ibm.task.api.AttributeInfo si l'EJB Human Task Manager est utilisé, peut être extraite de cet objet. Cette liste contient les noms et les types d'attribut des informations contenues dans les entités de l'ensemble de résultats. Elle contient également les méta-informations relatives aux attributs qui constituent la clé de ces entités.
entities	Une liste java.util.List d'objets com.ibm.bpe.api.Entity si l'EJB Business Flow Manager est utilisé, ou une liste d'objets com.ibm.task.api.Entity si l'EJB Human Task Manager est utilisé.
locale	L'environnement local calculé pour le paramètre système `$LOCALE`.

Les instances de la classe Entity contiennent les informations extraites de la requête lancée à partir de la table de requête. Une entité représente un objet identifiable de manière unique ; par exemple, une tâche, une instance de processus, une activité ou une escalade. Les propriétés suivantes sont disponibles pour les entités :

Tableau 51. Propriétés d'une entité renvoyée par l'API de table de requête

Propriété	Description
EntityInfo	L'objet EntityInfo qui est également contenu dans l'ensemble de résultats d'entités. Une liste java.util.List d'objets com.ibm.bpe.api.AttributeInfo si l'EJB Business Flow Manager est utilisé, ou une liste d'objets com.ibm.task.api.AttributeInfo si l'EJB Human Task Manager est utilisé, peut être extraite de cet objet. Cette liste contient les noms et les types d'attribut des informations contenues dans les entités de l'ensemble de résultats. Elle contient également les méta-informations relatives aux attributs qui constituent la clé de ces entités.
attributeValue (*nomAttribut*)	La valeur de l'attribut spécifié qui est extrait pour cette entité. Le type est contenu dans l'objet AttributeInfo correspondant de cet attribut.

Tableau 51. Propriétés d'une entité renvoyée par l'API de table de requête (suite)

Propriété	Description
attributeValuesOfArray (*nomAttribut*)	Un tableau de valeurs. Utilisez cette propriété si la valeur de la propriété *array* de l'objet AttributeInfo est 'true', ce qui est actuellement le cas uniquement si l'attribut fait référence à des données d'élément de travail.

Le nombre d'entités contenues dans l'ensemble de résultats est extrait à l'aide de la méthode size sur la liste d'entités.

Exemple : API de table de requête par entité :

```
...
// L'exemple suivant illustre une requête exécutée sur la
// table prédéfinie TASK et utilisant l'API par entité

...
// exécution de la requête
// le service est un objet (Local)BusinessFlowManager ou un
// objet (Local)HumanTaskManager
EntityResultSet rs = service.queryEntities("TASK", null, null, null);

// obtenir les méta-informations des entités
EntityInfo ei = rs.getEntityInfo();
List atts = ei.getAttributeInfo();

// obtenir les entités et itérer dessus
Iterator entitiesIter = rs.getEntities().iterator();
while (entitiesIter.hasNext()) {

      // manipuler une entité particulière
      Entity en = (Entity) entitiesIter.next();

      for (int i = 0; i < atts.size(); i++) {
            AttributeInfo ai = (AttributeInfo) atts.get(i);
            Serializable value = en.getAttributeValue(ai.getName()) ;

            // traiter...
      }
}
...
```

RowResultSet

Une instance de la classe com.ibm.bpe.api.RowResultSet est renvoyée par la méthode queryRows si l'EJB Business Flow Manager est utilisé. Une instance de la classe com.ibm.task.api.RowResultSet est renvoyée par la méthode queryRows si l'EJB Human Task Manager est utilisé. Ce type d'ensemble de résultats est similaire à un ensemble de résultats JDBC. Un ensemble de résultats de lignes a les propriétés suivantes :

Tableau 52. Propriétés d'un ensemble de résultats de lignes renvoyé par l'API de table de requête

Propriété	Description
primaryQueryTableName	• Si la requête a été exécutée sur une table de requête composite, cette propriété est le nom de la table de requête principale. • Si la requête a été exécutée sur une table de requête prédéfinie ou supplémentaire, cette propriété est le nom de la table en question ; autrement dit, elle a la même valeur que la propriété *queryTableName*.
attributeInfo	Cette propriété contient une liste d'objets com.ibm.bpe.api.AttributeInfo si l'EJB Business Flow Manager est utilisé, ou une liste d'objets com.ibm.task.api.AttributeInfo si l'EJB Human Task Manager est utilisé. Ces objets décrivent les méta-informations pour cet ensemble de résultats. Chaque objet AttributeInfo contient le nom et le type d'un attribut. Il ne contient pas de métadonnées relatives aux clés, car les ensembles de résultats de lignes n'ont pas de clé.
attributeValue	La valeur de l'attribut spécifié qui a été extrait pour cette ligne. Le type est contenu dans l'objet AttributeInfo correspondant de cet attribut.
next, first, last, previous	La navigation parmi les lignes de l'ensemble de résultats se fait au moyen de ces méthodes. Comparez leur utilisation à celle des itérateurs, des énumérations ou des ensembles de résultats JDBC.

Le nombre de lignes contenues dans l'ensemble de résultats relatif aux lignes est extrait à l'aide de la méthode size sur la liste de lignes.

Exemple : API de table de requête par ligne

```
...
// L'exemple suivant illustre une requête exécutée sur la
// table prédéfinie TASK et utilisant l'API par ligne
...
// exécution de la requête
// le service est un objet (Local)BusinessFlowManager ou un
// objet (Local)HumanTaskManager
RowResultSet rs = service.queryRows("TASK", null, null, null);

// obtenir les méta-informations des entités
List atts = rs.getAttributeInfo();

// obtenir les entités et itérer dessus
while (rs.next()) {

      // manipuler une ligne particulière
      for (int i = 0; i < atts.size(); i++) {
            AttributeInfo ai = (AttributeInfo) atts.get(i);
            Serializable value = rs.getAttributeValue(ai.getName()) ;

            // traiter...
      }
}
...
```

Requêtes sur des tables de requête pour l'extraction de métadonnées

L'exécution de requêtes sur les tables de requête définies dans Business Process Choreographer s'effectue au moyen de l'API de table de requête. Des méthodes sont disponibles pour permettre l'extraction de métadonnées des tables de requête.

Les méthodes suivantes sont disponibles pour l'extraction de métadonnées lorsque vous exécutez des requêtes sur les tables de requête dans Business Process Choreographer en utilisant l'API de table de requête :

Tableau 53. Méthodes pour l'extraction de métadonnées des tables de requête

Fonction	Méthode
Renvoyer les métadonnées d'une table de requête spécifique	getQueryTableMetaData
Renvoyer la liste des métadonnées de table de requête ayant des propriétés spécifiques	findQueryTableMetaData
Renvoyer le contenu d'une table de requête en fonction d'entités et d'un sous-ensemble des métadonnées des attributs sélectionnés	queryEntities
Renvoyer le contenu d'une table de requête en fonction de lignes et d'un sous-ensemble des métadonnées des attributs sélectionnés	queryRows

Les métadonnées des tables de requête comprennent des informations en rapport avec la structure des tables et d'autres informations en rapport avec l'internationalisation.

Le tableau suivant présente les métadonnées relatives à la structure d'une table de requête.

Tableau 54. Métadonnées relatives à la structure d'une table de requête

Métadonnées	Description	Renvoyées par getQuery-TableMetaData	Renvoyées par findQuery-TableMetaData	Renvoyées par queryEntities	Renvoyées par queryRows
Nom de la table de requête	Le nom de la table de requête	Oui	Oui	Oui	Oui
Nom de la table de requête principale	Le nom de la table de requête lorsqu'il s'agit d'une table prédéfinie ou supplémentaire, ou le nom de la table principale lorsque la table interrogée est une table de requête composite	Oui	Oui	Oui	Oui
Type	Le type de table de requête : composite, prédéfinie ou supplémentaire.	Oui	Oui	Non	Non
Autorisation	L'autorisation qui est définie sur la table de requête : • Utilisation d'éléments de travail • Autorisation par instance ou par rôle ou absence de contrôle d'autorisation	Oui	Oui	Non	Non

Tableau 54. Métadonnées relatives à la structure d'une table de requête (suite)

Métadonnées	Description	Renvoyées par getQuery-TableMetaData	Renvoyées par findQuery-TableMetaData	Renvoyées par queryEntities	Renvoyées par queryRows
Attributs définis	Les métadonnées des attributs qui sont définis sur la table de requête	Oui	Oui	Non. Les métadonnées des attributs sélectionnés sont renvoyées.	Non. Les métadonnées des attributs sélectionnés sont renvoyées.
Attributs de clé	Les attributs de clé de la table de requête	Oui	Oui	Oui	Non. Ne concerne pas les requêtes par ligne.

Le tableau suivant présente les métadonnées relatives à l'internationalisation d'une table de requête.

Tableau 55. Métadonnées relatives à l'internationalisation d'une table de requête

Métadonnées	Description	Renvoyées par getQuery-TableMetaData	Renvoyées par findQuery-TableMetaData	Renvoyées par queryEntities	Renvoyées par queryRows
locales[]	Environnements locaux pour lesquels les noms d'affichage et les descriptions de la table de requête et des attributs sont définis.	Oui	Oui	Non	Non
Environnement local	Valeur du paramètre système `$LOCALE` qui résulte de l'environnement local qui est transmis à l'API.	Oui	Oui	Oui	Oui
Nom d'affichage et description de la table de requête	Les noms d'affichage et les descriptions de la table de requête, fournis pour tous les environnements locaux définis.	Oui	Oui	Non	Non
Noms d'affichage et descriptions des attributs	Les noms d'affichage et les descriptions des attributs, fournis pour tous les environnements locaux définis.	Oui	Oui	Non	Non

Toutes les méthodes de l'API de table de requête EJB qui renvoient des métadonnées de table de requête acceptent un paramètre d'environnement local ; par exemple, `FilterOptions.setLocale` et `MetaDataOptions.setLocale`. Ce paramètre doit recevoir pour valeur l'environnement local Java que le client utilise pour présenter les informations à l'utilisateur. Ce paramètre d'environnement local sert à calculer la valeur du paramètre système `$LOCALE`, lequel peut être utilisé dans les filtres et les critères de sélection. L'environnement local renvoyé contient l'environnement local Java qui est utilisé pour `$LOCALE`.

Si votre requête extrait les noms d'affichage et les descriptions d'une table de requête spécifique, passez getLocale aux méthodes concernées afin d'obtenir ces noms et ces descriptions dans le même environnement local que celui des descriptions des tâches. Par exemple, les descriptions suivantes sont attachées au moyen du critère de sélection `'LOCALE=$LOCALE'`.

Exemple

```
// l'exemple suivant montre comment extraire
// les métadonnées d'une table de requête composite particulière

...
// exécution de la requête
MetaDataOptions mdo = new MetaDataOptions("TASK", null, false, new Locale("en_US"));
List list = bfm.findQueryTableMetaData(mdo);

// pour obtenir les métadonnées d'une table de requête spécifique
// utiliser bfm.getQueryTableMetaData(...)

// itérer sur la liste des tables de requête dont la table principale est TASK
// => au moins une table de requête est renvoyée : la table prédéfinie TASK

Iterator iter = list.iterator();
while( iter.hasNext() ) {
 QueryTableMetaData md = (QueryTableMetaData) iter.next();
 Locale effectiveLocale = md.getLocale();
 String queryTableDisplayName = md.getDisplayName(effectiveLocale);
 System.out.println("found query table: " + queryTableDisplayName);
 List attributesList = md.getAttributeMetaData();
 Iterator attrIter = attributesList.iterator();
 while (attrIter.hasNext()) {
  AttributeMetaData amd = (AttributeMetaData) attrIter.next();
  String attributeDisplayName = amd.getDisplayName(effectiveLocale);
  System.out.println("\tattribute:" + attributeDisplayName);
 }
}
```

Environnement local le plus proche

Lors de la spécification des conditions sur une table de requête associée, l'utilisation de la valeur $LOCALE peut renvoyer des résultats inattendus si l'environnement local spécifié ne correspond pas exactement aux métadonnées. Par exemple, si vous transmettez l'environnement local en_US avec une requête sur une table de requête qui contient des métadonnées spécifiant la langue en, le résultat renvoyé est null.

Pour éviter cela, vous pouvez utiliser LOCALE=$LOCALE_BEST_MATCH, qui applique un algorithme d'environnement local le plus proche afin de déterminer le véritable environnement local utilisé dans la requête. Par exemple, une requête avec l'environnement local en_US sur une table de requête en langue en est effectuée en utilisant en.

Vous ne pouvez pas spécifier d'autres opérateurs logiques ou de comparaison dans la condition LOCALE=$LOCALE_BEST_MATCH. Vous ne pouvez utiliser cet algorithme d'environnement local le plus proche que sur les tables de requête associée, en le spécifiant comme condition sur les autres résultats de requête dans une erreur.

Internationalisation pour les métadonnées des tables de requête

L'internationalisation est prise en charge pour les métadonnées des tables de requête.

Les noms d'affichage et les descriptions des tables de requête composites peuvent être fournis en différentes langues correspondant à différents codes d'environnement local. Par exemple, il est possible de définir, pour une table de requête composite, un nom d'affichage en anglais (environnement local en_US), en allemand (environnement de) et dans la langue de l'environnement local par défaut. Cette configuration est réalisée durant le développement de la table de

requête, à l'aide de Query Table Builder. Pour déployer une table de requête avec des noms d'affichage et des descriptions en différentes langues, vous devez utiliser l'option `-deploy jarFile` lorsque vous déployez la table dans le conteneur Business Process Choreographer.

En termes de traitement des environnements locaux, le comportement des méthodes de l'API de table de requête, queryEntities et queryRows, et des méthodes de manipulation des métadonnées fournies par cette même API, getQueryTableMetaData et findQueryTableMetaData, est similaire à celui qui est obtenu avec les regroupements de ressources Java.

Pour que les noms d'affichage et les descriptions des métadonnées d'une table de requête restent cohérents avec le contenu de ladite table, la valeur du paramètre système `$LOCALE` dépend des environnements locaux pour lesquels les noms d'affichage et les descriptions sont spécifiés dans la table de requête.

Exemple

Dans le scénario suivant, un client affiche des listes de tâches et des listes de processus et crée une demande pour interroger une table de requête.

- Le client n'a pas spécifié d'environnement local particulier à utiliser pour présenter l'information à l'utilisateur. Il est probable que l'application n'est pas conçue pour fonctionner en différentes langues.
  - Un environnement local par défaut est spécifié pour les noms d'affichage et les descriptions des tables de requête. C'est le cas pour toutes les tables composites et supplémentaires construites avec la version actuelle de Query Table Builder. Par conséquent la valeur de `$LOCALE` est `default`.
  - Pour l'environnement local par défaut, aucun nom d'affichage ou description n'est spécifié sur la table de requête. C'est le cas pour toutes les tables de requête prédéfinies ainsi que pour toutes les tables de requête déployées avec l'option `-deploy qtdFile`. La valeur de `$LOCALE` est basée sur la méthode du regroupement de ressources Java.
- Le client a spécifié l'environnement local à utiliser pour présenter l'information à l'utilisateur. Par exemple, c'est le cas lorsque l'API REST pour les tables de requête est utilisée.
  - Les noms d'affichage et les descriptions sont spécifiés sur la table de requête. La méthode du regroupement de ressources Java est utilisée pour calculer la valeur de `$LOCALE` d'après l'environnement local qui est transmis par le client.
  - Les noms d'affichage et les descriptions ne sont pas spécifiés sur la table de requête. La variable `$LOCALE` est réglée à la valeur qui est transmise par le client.

Environnement local le plus proche

Lors de la spécification des conditions sur une table de requête associée, l'utilisation de la valeur `$LOCALE` peut renvoyer des résultats inattendus si l'environnement local spécifié ne correspond pas exactement aux métadonnées. Par exemple, si vous transmettez l'environnement local `en_US` avec une requête sur une table de requête qui contient des métadonnées spécifiant la langue `en`, le résultat renvoyé est `null`.

Pour éviter cela, vous pouvez utiliser `LOCALE=$LOCALE_BEST_MATCH`, qui applique un algorithme d'environnement local le plus proche afin de déterminer le véritable

environnement local utilisé dans la requête. Par exemple, une requête avec l'environnement local `en_US` sur une table de requête en langue `en` est effectuée en utilisant `en`.

Vous ne pouvez pas spécifier d'autres opérateurs logiques ou de comparaison dans la condition `LOCALE=$LOCALE_BEST_MATCH`. Vous ne pouvez utiliser cet algorithme d'environnement local le plus proche que sur les tables de requête associée, en le spécifiant comme condition sur les autres résultats de requête dans une erreur.

Tâches associées

«Création de tables de requête pour Business Process Choreographer Explorer», à la page 417
Vous pouvez utiliser des tables de requête à la place de l'API query des EJB pour améliorer les performances de Business Process Choreographer Explorer. Pour créer les tables de requête, utilisez Query Table Builder.

Tables de requête et performances des requêtes

Les tables de requête offrent un nouveau modèle de programmation propre, conçu pour le développement d'applications clientes qui extraient des listes de tâches manuelles et de processus métier dans Business Process Choreographer. Le recours aux tables de requête améliore les performances. Des informations sont fournies sur les paramètres d'API de table de requête et les autres facteurs ayant un impact sur les performances.

Les temps de réponse des requêtes exécutées sur les tables de requête dépendent principalement des options d'autorisation, des filtres et des critères de sélection utilisés. Voici une liste de points à prendre en considération lorsqu'il est question de performances.

- Les options d'autorisation influent de façon importante sur les performances. Activez l'autorisation en utilisant le moins possible d'options et utilisez principalement les éléments de travail de personne et de groupe. Evitez d'utiliser des éléments de travail inherited. Les options d'autorisation peuvent être restreintes de manière plus importante au moment de l'exécution de la requête. En outre, si cela n'est pas nécessaire, indiquez que l'autorisation par le biais des éléments de travail n'est pas requise.
- Si une autorisation par éléments de travail est requise, définissez un filtre d'autorisation. Par exemple, pour autoriser uniquement les objets de la table de requête qui contiennent un élément de travail de propriétaire potentiel, utilisez WI.REASON=REASON_POTENTIAL_OWNER.
- Le filtrage de la table de requête principale est par exemple utile pour autoriser uniquement les tâches à l'état Prêt de la table de requête lorsque TASK est la table de requête principale.
- En termes de performances, les filtres de table de requête et les filtres de requête (filtres transmis à l'exécution de la requête) sont moins efficaces que les filtres principaux.
- Lorsque cela est possible, évitez d'utiliser des paramètres dans les filtres et les critères de sélection.
- Evitez également d'utiliser des opérateurs `LIKE` dans les filtres et les critères de sélection.

Définition des tables de requête composites

Le tableau suivant décrit l'impact qu'ont les options définies pour les tables de requête composites sur les performances des requêtes. Il traite également d'autres sujets en rapport avec les définitions des tables de requête composites. L'indication

figurant dans la colonne Impact sur les performances est une moyenne ; les impacts réellement observés peuvent varier.

Tableau 56. Les options applicables aux tables de requête composites et leur impact sur les performances des requêtes

Objet ou sujet	Impact sur les performances	Description
Filtre de table de requête	Négatif	Les filtres appliqués aux tables de requête sont ceux qui ont l'impact le plus négatif sur les performances des requêtes. Généralement, ils ne peuvent pas utiliser les index définis dans la base de données.
Filtre de la table de requête principale	Positif	Un filtre appliqué à la table principale intervient très tôt dans le calcul des résultats de la requête et se révèle donc particulièrement efficace en termes de performances. Pour cette raison, il est conseillé de restreindre le contenu de la table de requête au moyen d'un filtre appliqué à la table de requête principale.
Filtre d'autorisation	Positif	Un filtre d'autorisation peut améliorer les performances de la requête dans des proportions comparables à celles du filtre appliqué à la table de requête principale. Dans la mesure du possible, un filtre d'autorisation doit être appliqué. Par exemple, si les éléments de travail reader (lecteur) ne doivent pas être pris en compte, spécifiez `WI.REASON=REASON_READER`.
Critères de sélection	Aucun	Pour certaines relations entre la table principale et les tables attachées, il faut définir un critère de sélection afin de garantir que ces relations sont bien du type un à un ou un à zéro. Généralement, un critère de sélection n'a qu'un impact légèrement négatif sur les performances, car il est évalué pour un petit nombre de lignes uniquement.
Paramètres	Aucun	Actuellement, l'emploi de paramètres dans les tables de requête n'a pas d'impact négatif sur les performances. Ils ne doivent cependant être utilisés qu'en cas de nécessité.
Autorisation par instance	Négatif	Si l'autorisation par instance est utilisée, l'existence d'un élément de travail doit être vérifiée pour chaque objet dans la table de requête. Les éléments de travail sont représentés par les entrées de la table de requête WORK_ITEM. Cette vérification affecte les performances.
Autorisation par instance : • everybody • individuals • groups • inherited	Négatif	Chaque type d'élément de travail dont l'utilisation est spécifiée dans la table de requête a un impact sur les performances. Les applications ayant à soumettre d'importants volumes de requêtes doivent, si possible, se limiter à l'emploi des éléments de travail 'individuals' et 'groups', voire à un seul de ces éléments. Les éléments de travail 'inherited' ne sont généralement pas nécessaires, en particulier pour la définition de listes de tâches renvoyant des tâches manuelles qui représentent des tâches à effectuer. Ils doivent en revanche être utilisés lorsqu'il est clair qu'ils sont indispensables ; par exemple, pour retourner des listes de tâches qui appartiennent à un processus métier, si une personne, compte tenu de son autorisation, ne bénéficie que d'un accès en lecture au processus métier englobant les tâches en question.

Tableau 56. Les options applicables aux tables de requête composites et leur impact sur les performances des requêtes (suite)

Objet ou sujet	Impact sur les performances	Description
Autorisation par rôle ou absence de contrôle d'autorisation	Aucun	Si l'autorisation par rôle est utilisée, ou si aucun contrôle d'autorisation n'est exercé, les vérifications par rapport aux éléments de travail n'ont pas lieu.
Nombre d'attributs définis	Actuellement aucun	Le nombre d'attributs contenus dans une table de requête n'a pas d'incidence sur les performances. Néanmoins, seuls les attributs véritablement nécessaires doivent faire partie d'une table de requête.

API de table de requête

Le tableau suivant décrit l'impact qu'ont les options définies pour l'API de table de requête sur les performances des requêtes. L'indication figurant dans la colonne Impact sur les performances est une moyenne ; les impacts réellement observés peuvent varier.

Tableau 57. Les options de l'API de table de requête et leur impact sur les performances des requêtes

Option	Impact sur les performances	Description
Attributs sélectionnés	Négatif (moins il y en a, mieux c'est)	Le nombre d'attributs sélectionnés lorsqu'une requête est exécutée sur une table de requête détermine directement le nombre de données qui devront être traitées à la fois par la base de données et par l'environnement d'exécution de table de requête de Business Process Choreographer. De plus, pour les tables de requête composites, les informations des tables attachées n'ont besoin d'être extraites que si elles sont spécifiées par les attributs sélectionnés ou référencées par le filtre de table de requête ou par le filtre de requête.
Filtre de requête	Négatif	S'il est spécifié, le filtre de requête a le même impact sur les performances que le filtre de table de requête. Cependant, il est préférable de spécifier les filtres sur les tables de requête plutôt que de les passer à l'API de table de requête.
Attributs de tri	Négatif	Le tri des résultats d'une requête est coûteux en temps de traitement ; de plus, dès lors que le tri est utilisé, les optimisations de la base de données sont restreintes. Le tri doit être évité s'il n'est pas indispensable. La plupart des applications en ont toutefois besoin.
Seuil	Positif	La spécification d'un seuil peut améliorer significativement les performances des requêtes. Il est recommandé de toujours spécifier un seuil.
Nombre d'éléments à sauter (skipCount)	Négatif	Le saut d'un nombre particulier d'objets dans l'ensemble de résultats d'une requête est coûteux en temps de traitement ; il ne doit être réalisé qu'en cas d'absolue nécessité (par exemple, pour la pagination des résultats).
Fuseau horaire	Aucun	Le réglage de fuseau horaire n'a pas d'incidence sur les performances.

Tableau 57. Les options de l'API de table de requête et leur impact sur les performances des requêtes (suite)

Option	Impact sur les performances	Description
Environnement local	Aucun	Le réglage d'environnement local n'a pas d'incidence sur les performances.
Lignes distinctes	Négatif	L'emploi de l'opérateur 'distinct' dans les requêtes a un certain impact sur les performances, mais il est parfois incontournable pour éviter l'extraction de lignes en double. Cette option concerne uniquement les requêtes par ligne ; elle est ignorée dans les autres cas.
Requêtes de comptage	Positif	Si seul le nombre total d'entités ou le nombre total de lignes d'une requête particulière est nécessaire (autrement dit, s'il n'est pas nécessaire d'extraire le contenu de toutes les entrées de la table de requête), il convient d'utiliser la méthode queryEntityCount ou la méthode queryRowCount, respectivement. L'environnement d'exécution de Business Process Choreographer peut appliquer des optimisations valides uniquement pour les requêtes de comptage.

Autres considérations

Les autres facteurs à prendre en considération en ce qui concerne les performances sont les suivants :

Tableau 58. Performances des tables de requête - Autres considérations

Elément	Description
Nombre de tables de requête sur le système	Le nombre de tables de requête déployées dans un conteneur Business Process Choreographer n'influe pas sur les performances des requêtes exécutées sur ces tables. Il est également sans conséquence sur la navigation des instances de processus métier et n'a pas non plus d'impact sur les opérations de réclamation et d'achèvement des tâches manuelles. Maintenez-le quand même à un niveau raisonnable, sous peine de compliquer la maintenance de votre environnement. Généralement, une table de requête particulière représente une seule et même liste de tâches ou liste de processus affichée dans l'interface utilisateur.

Tableau 58. Performances des tables de requête - Autres considérations (suite)

Elément	Description
Réglage de la base de données	Même si l'accès au contenu des tables de requête est réalisé au moyen de code SQL optimisé, le réglage des bases de données doit être implémenté dans une base de données Business Process Choreographer : • La mémoire réservée à la base de données doit être réglée au maximum, en tenant compte des besoins en mémoire des autres processus exécutés sur le serveur de base de données et des limites propres au matériel. • Les statistiques relatives à la base de données doivent être le plus à jour possible ; elles doivent donc être actualisées à intervalles réguliers. Généralement, les procédures à cet effet sont déjà implémentées dans les grandes topologies. Par exemple, prévoyez une collecte hebdomadaire des statistiques destinées à l'optimiseur pour refléter les changements des données dans la base de données. • Les systèmes de gestion de bases de données fournissent des outils qui permettent de réorganiser (ou défragmenter) les conteneurs de données. L'agencement physique des données dans la base de données peut aussi influer sur les performances des requêtes et les voies d'accès qu'elles empruntent. • L'optimisation des index est l'un des facteurs clés en matière de performance des requêtes. Business Process Choreographer est fourni avec des index prédéfinis, optimisés pour conférer les meilleures performances à la navigation des processus et aux requêtes dans des scénarios types. Dans les environnements personnalisés, d'autres index peuvent être nécessaires afin de soutenir les requêtes sur les listes de tâches ou de processus qui génèrent ou manipulent de gros volumes de données. Utilisez les outils fournis par la base de données pour prendre en charge les requêtes exécutées sur une table de requête.

Création de tables de requête pour Business Space

Dans l'outil Query Table Builder, vous pouvez utiliser la définition de table de requête composite contenant des propriétés prédéfinies en vue de créer des tables de requête pour Business Space.

task_prereq

L'outil Query Table Builder, disponible sous forme de plug-in Eclipse, peut être téléchargé à partir du site des SupportPacs WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

task_procedure

1. Dans Query Table Builder, cliquez avec le bouton droit de la souris sur votre projet, puis sélectionnez **Nouveau → Composite Query Definition for Business Space**. Suivez les instructions de l'assistant pour créer une définition de table de requête. La nouvelle définition de table de requête est constituée des propriétés prédéfinies. Si nécessaire, ajoutez d'autres propriétés à la définition de table de requête et déployez le fichier de définition de table de requête sur WebSphere Process Server.

Remarque : Les noms que vous attribuez aux propriétés dans l'outil Query Table Builder sont utilisés pour les propriétés de tâche dans Business Space for Choreographer.

2. Après avoir créé et déployé les fichiers de définition de table de requête vous pouvez les configurer dans Business Space. Par exemple, si vous avez déployé un fichier de définition de table pour le widget Liste des tâches :
   a. Ouvrez le menu du widget et sélectionnez **Configurer**, puis l'onglet **Contenu**.
   b. Dans l'onglet **Contenu** ouvrez la liste déroulante **Sélectionnez la liste de tâches à afficher** pour afficher les listes que vous pouvez rendre accessibles à l'utilisateur du widget. Sélectionnez **Ajouter des listes de tâches**. La définition de table de requête que vous avez déployée doit être disponible pour la sélection dans cette liste.

   Si la définition de table de requête n'est pas disponible vous devez revenir à l'outil Query Table Builder et vérifier si le fichier de définition a été correctement défini et déployé.

Création de tables de requête pour Business Process Choreographer Explorer

Vous pouvez utiliser des tables de requête à la place de l'API query des EJB pour améliorer les performances de Business Process Choreographer Explorer. Pour créer les tables de requête, utilisez Query Table Builder.

task_prereq

L'outil Query Table Builder, disponible sous forme de plug-in Eclipse, peut être téléchargé à partir du site des SupportPacs WebSphere Business Process Management. Recherchez le SupportPac PA71 WebSphere Process Server - Query Table Builder. Pour accéder au lien, consultez la section consacrée aux références de cette rubrique.

task_procedure

1. Dans Query Table Builder, cliquez avec le bouton droit de la souris sur votre projet, puis sélectionnez **Nouveau → Composite Query Definition for Business Space**. Cette option que toutes les colonnes requises pour Business Process Choreographer Explorer sont présélectionnées.
2. Suivez les instructions de l'assistant pour créer une définition de table de requête. Si nécessaire, ajoutez d'autres propriétés à la définition de table de requête. Tenez compte des aspects suivants lorsque vous définissez votre table de requête :

   Critères de filtrage

   Lorsque vous créez des vues dans Business Process Choreographer Explorer basées sur des tables de requête, vous ne pouvez pas spécifier de filtres ou de variables supplémentaires pour vos critères de recherche. Vous devez spécifier ces critères de filtrage et les paramètres des variables lorsque vous créez la table de requête.

   Vous pouvez utiliser une table de requête pour plusieurs vues dans Business Process Choreographer Explorer à l'aide des paramètres de la définition de table de requête. Pour une plus grande flexibilité, vous pouvez également indiquer si les valeurs par défaut des paramètres peuvent être remplacées lorsque la requête de la vue personnalisée est exécutée.

Autorisation

Lorsque vous créez des vues dans Business Process Choreographer Explorer basées sur des tables de requête, vous ne pouvez pas filtrer vos critères de recherche en fonction du rôle utilisateur. Vous devez définir les critères de filtrage des rôles utilisateur lorsque vous définissez la table de requête. Pour les tables de requête principales basées sur des données de modèle, utilisez une autorisation basée sur les instances et non une autorisation basée sur les rôles, comme type d'autorisation. Pour les tables de requête principales basées sur des données d'instance, spécifiez le filtre d'autorisation basée sur les instances approprié.

Internationalisation

Lorsque vous définissez des propriétés dans l'outil Query Table Builder, vous pouvez également spécifier les noms et les descriptions de ces propriétés dans des langues différentes. Lorsque la requête de la vue personnalisée est exécutée, Business Process Choreographer Explorer utilise la traduction appropriée pour le paramètre linguistique de votre navigateur.

Nom d'affichage et description de la définition de table de requête

Dans l'outil Query Table Builder, vous pouvez fournir un nom d'affichage et une description pour toutes les langues prises en charge par la vue.

Nom d'affichage et description des colonnes

Lors de l'exécution, Business Process Choreographer Explorer extrait les noms de colonne internationalisés appropriés qui sont affichés dans une liste de résultats. Pour les colonnes provenant de votre table de requête principale, telles que PIID, Business Process Choreographer Explorer utilise les traductions déjà disponibles pour toutes les langues prises en charge.

Pour les colonnes provenant d'une table de requête associée, telle que QUERY_PROPERTY, vous devez fournir les noms d'affichage et les descriptions dans l'outil Query Table Builder, dans toutes les langues prises en charge par votre entreprise.

Noms et descriptions des tâches

Si vous avez des noms et des descriptions de tâche internationalisés dans WebSphere Integration Developer, ils sont affichés dans Business Process Choreographer Explorer en fonction des paramètres de langue et de pays de votre navigateur. Si les paramètres de votre navigateur ne correspondent pas aux paramètres définis dans le modèle de processus, la traduction de la langue par défaut est utilisée.

Critères de tri

Lorsque vous définissez des critères de tri pour une définition de table de requête, sachez que plusieurs propriétés (par exemple, l'état du processus) sont stockées sous forme d'entiers, tandis que Business Process Choreographer Explorer les affiche comme chaînes traduites dans la liste résultante. Des résultats inattendus peuvent donc être générés dans certaines langues.

La nouvelle définition de table de requête est constituée des propriétés prédéfinies et des propriétés supplémentaires que vous définissez.

task_postreq

Déployez et testez la définition de requête, dans l'outil Query Table Builder, sur un serveur d'applications. S'il s'agit du serveur auquel Business Process Choreographer Explorer est connecté, vous pouvez maintenant utiliser la table de requête lorsque vous personnalisez Business Process Choreographer Explorer en fonction de votre utilisation ou pour des groupes d'utilisateurs différents. Si Business Process Choreographer Explorer est connecté à un autre serveur, vous devez déployer la table de requête sur le serveur approprié pour pouvoir l'utiliser pour créer des vues personnalisées.

Concepts associés

«Autorisation pour les tables de requête», à la page 383
Vous pouvez utiliser une autorisation par instance, une autorisation par rôle ou aucune autorisation lorsque vous exécutez des requêtes ou des tables de requête.

«Filtres et critères de sélection des tables de requête», à la page 378
Les filtres et les critères de sélection sont définis pendant la phase de développement des tables de requête, à l'aide de l'outil Query Table Builder, qui utilise une syntaxe similaire aux clauses SQL WHERE. En définissant clairement des filtres et des critères de sélection, vous pouvez spécifier des conditions basées sur les attributs des tables de requête.

«Internationalisation pour les métadonnées des tables de requête», à la page 410
L'internationalisation est prise en charge pour les métadonnées des tables de requête.

API de requête EJB de Business Process Choreographer

Les méthodes query ou queryAll du service API vous permettent d'extraire des informations stockées relatives aux processus métier et aux tâches.

La méthode query peut être appelée par tous les utilisateurs, elle renvoie les propriétés des objets pour lesquels les éléments de travail existent. La méthode queryAll peut être appelée uniquement par les utilisateurs disposant de l'un des rôles Java EE suivants : BPESystemAdministrator, TaskSystemAdministrator, BPESystemMonitor ou TaskSystemMonitor. Cette méthode renvoie les propriétés de tous les objets qui sont stockés dans la base de données.

Toutes les requêtes API sont mappées avec les requêtes SQL. La forme de la requête SQL résultante dépend de ce qui suit :

- Si la requête a été appelée par une personne ayant l'un des rôles Java EE.
- Les objets qui sont interrogés. Des vues prédéfinies des bases de données sont disponibles pour vous permettre de rechercher les propriétés de l'objet.
- L'insertion d'une clause From, de conditions de jointure et de conditions propres à l'utilisateur pour le contrôle d'accès.

Les requêtes peuvent inclure à la fois des propriétés personnalisées et des propriétés de variable. Si vous ajoutez plusieurs propriétés personnalisées ou propriétés de variables à votre requête, des jointures automatiques sont créées dans la table de base de données correspondante. Suivant le système de base de données utilisé, les appels de `query()` peuvent avoir des implications diverses sur les performances.

Vous pouvez également stocker des requêtes dans la base de données Business Process Choreographer à l'aide de la méthode createStoredQuery. Vous fournissez les critères de requête lors de la définition de la requête stockée. Les critères sont appliqués lors de l'exécution de la requête stockée, ce qui signifie que les données

sont regroupées durant cette période. Si la requête stockée contient des paramètres, ils sont également résolus lors de son exécution.

Pour plus d'informations sur les interfaces API de Business Process Choreographer, consultez Javadoc dans le package com.ibm.bpe.api pour les méthodes relatives aux processus et dans le package com.ibm.task.api pour les méthodes relatives aux tâches.

Concepts associés

«Syntaxe de la méthode query dans l'API»
La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

«Conditions d'accès propres à l'utilisateur», à la page 427
Les conditions d'accès propres à l'utilisateur sont ajoutées lorsque l'instruction SQL SELECT est générée par la requête API. Ces conditions garantissent que seuls ces objets sont renvoyés à l'appelant parce que conformes à la condition spécifiée par l'appelant et rendus accessibles à ce dernier.

«Exemples de méthodes query et queryAll», à la page 429
Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Tâches associées

«Gestion des requêtes stockées», à la page 433
Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

Syntaxe de la méthode query dans l'API

La syntaxe des requêtes de l'API du Business Process Choreographer est similaire à celle des requêtes SQL. Une requête peut inclure les clauses Select, Where et Order-by ainsi que les paramètres Skip-tuples, Threshold et Time-zone.

La syntaxe de la requête dépend du type d'objet. Le tableau suivant présente la syntaxe correspondant aux différents types d'objet.

Tableau 59. Syntaxe de la requête pour les différents types d'objet

Objet	Syntaxe
Modèle de processus	`ProcessTemplateData[] queryProcessTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
Modèle de tâche	`TaskTemplate[] queryTaskTemplates (java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer threshold, java.util.TimeZone timezone);`
Données relatives aux processus métier et aux tâches	`QueryResultSet query (java.lang.String selectClause, java.lang.String whereClause, java.lang.String orderByClause, java.lang.Integer skipTuples java.lang.Integer threshold, java.util.TimeZone timezone);`

Concepts associés

«Clause Select»
La clause SELECT de la fonction identifie les propriétés de l'objet qui doivent être renvoyées par une requête.

«Clause Where», à la page 422
La clause WHERE de la fonction de requête décrit les critères de filtrage à appliquer au domaine de la requête.

«Clause Order-by», à la page 423
La clause ORDER BY de la fonction de requête spécifie les critères de tri pour l'ensemble de résultats de la requête.

«Paramètre Skip-tuples», à la page 424
Le paramètre skip-tuples spécifie le nombre de tuples dans l'ensemble de résultats de requête, en partant du début, à ignorer et à ne pas renvoyer à l'appelant dans l'ensemble des résultats de requête.

«Paramètre Threshold», à la page 424
Le paramètre threshold de la fonction de requête restreint le nombre d'objets renvoyés du serveur au client dans l'ensemble de résultats de requête.

«Paramètre Timezone», à la page 425
Le paramètre time-zone de la fonction de requête définit le fuseau horaire des constantes d'horodatage de la requête.

«Résultats d'interrogation», à la page 426
Un ensemble de résultats de requête contient les résultats d'une requête d'API de Business Process Choreographer.

Tâches associées

«Filtrage de données à l'aide de variables définies dans des requêtes», à la page 425
Un résultat de requête renvoie l'objet répondant aux critères de la recherche. Vous pouvez filtrer ces résultats selon les valeurs des variables.

Clause Select :

La clause SELECT de la fonction identifie les propriétés de l'objet qui doivent être renvoyées par une requête.

La clause SELECT décrit le résultat de la requête. Cette clause spécifie une liste de noms identifiant les propriétés des objets (colonnes du résultat) à renvoyer. Sa syntaxe est identique à celle de la clause SELECT de SQL ; utilisez la virgule pour séparer les différentes parties de la clause. Chaque partie de la clause doit spécifier une colonne d'une des vues prédéfinies. Les colonnes doivent être entièrement spécifiées par le nom de la vue et le nom de la colonne. Les colonnes renvoyées dans l'objet QueryResultSet sont affichées dans le même ordre que les colonnes spécifiées dans la clause Select.

La clause SELECT ne prend pas en charge des fonctions d'agrégation SQL telles AVG(), SUM(), MIN() ou MAX().

Pour sélectionner les propriétés de plusieurs paires nom-valeur, telles que des propriétés personnalisées ou des propriétés de variables pouvant être interrogées, ajoutez un compteur à un chiffre au nom de la vue. Ce compteur peut adopter une valeur comprise de 1 à 9.

Exemples de clauses SELECT

- "WORK_ITEM.OBJECT_TYPE, WORK_ITEM.REASON"

Obtient les type des objets associés et les motifs d'attribution des éléments de travail.

- "DISTINCT WORK_ITEM.OBJECT_ID"

  Obtient tous les ID des objets, sans les doublons, pour lesquels l'appelant a un élément de travail.
- "ACTIVITY.TEMPLATE_NAME, WORK_ITEM.REASON"

  Obtient les noms des activités pour lesquelles l'appelant a des éléments de travail, ainsi que leurs motifs d'attribution.
- "ACTIVITY.STATE, PROCESS_INSTANCE.STARTER"

  Obtient les états des activités et les initiateurs des instances de processus y associés.
- "DISTINCT TASK.TKIID, TASK.NAME"

  Obtient tous les ID et les noms de tâches, sans les doublons, pour lesquels l'appelant a un élément de travail.
- "TASK_CPROP1.STRING_VALUE, TASK_CPROP2.STRING_VALUE"

  Obtient les valeurs des propriétés personnalisées qui sont spécifiées dans la clause WHERE.
- "QUERY_PROPERTY1.STRING_VALUE, QUERY_PROPERTY2.INT_VALUE

  Extrait les valeurs des propriétés de variables pouvant être interrogées. Ces parties sont ensuite spécifiées dans la clause Where.
- "COUNT(DISTINCT TASK.TKIID)"

  Compte le nombre de éléments de travail pour les tâches uniques qui satisfont la clause WHERE.

Clause Where :

La clause WHERE de la fonction de requête décrit les critères de filtrage à appliquer au domaine de la requête.

La syntaxe de la clause Where est identique à celle de la clause SQL WHERE. Vous n'avez pas besoin d'ajouter explicitement une clause SQL à partir d'une clause ou des prédicats de jointure à la clause Where de l'API, car ces constructions sont ajoutées automatiquement lors de l'exécution de la requête. Si vous ne désirez pas appliquer de critères de filtre, spécifiez `null` comme valeur de la clause WHERE.

La syntaxe de la clause WHERE prend en charge :

- Mots clés : AND, OR, NOT
- Opérateurs de comparaison : =, <=, <, <>, >,>=, LIKE

  L'opération LIKE prend en charge les caractères génériques définis pour la base de données interrogée.
- Opération SET : IN

Les règles suivantes s'appliquent également :

- Spécifiez les constantes ID d'objet comme `ID('string-rep-of-oid')`.
- Spécifiez les constantes binaires comme `BIN('UTF-8 string')`.
- Utilisez des constantes symboliques au lieu des énumérations d'entiers. Par exemple, au lieu de spécifier une expression d'état d'activité`ACTIVITY.STATE=2`, spécifiez `ACTIVITY.STATE=ACTIVITY.STATE.STATE_READY`.
- Si la valeur de la propriété de l'instruction de comparaison contient des guillemets simples ('), doublez ces guillemets ; par exemple, `"TASK_CPROP.STRING_VALUE='d''automatisation'"`.

- Faites référence aux propriétés de plusieurs paires nom-valeur, telles que des propriétés personnalisées, en ajoutant un suffixe à un chiffre au nom de la vue. Par exemple : `"TASK_CPROP1.NAME='prop1' AND "TASK_CPROP2.NAME='prop2'"`
- Spécifiez les constantes d'horodatage comme `TS('yyyy-mm-ddThh :mm :ss')`. Pour faire référence à la date actuelle, spécifiez `CURRENT_DATE` comme horodatage.

  Au moins une valeur de date ou d'heure doit être spécifiée dans l'horodatage.
  - Si vous spécifiez uniquement une date, la valeur de l'heure sera zéro.
  - Si vous spécifiez uniquement une heure, la valeur de la date sera la date actuelle.
  - Si vous spécifiez une date, l'année doit consister d'au moins quatre chiffres ; les valeurs du mois et du jour sont optionnelles. Les valeurs du jour et du mois manquantes seront remplacées par 01. Par exemple, `TS('2003')` et identique à `TS('2003-01-01T00 :00 :00')`.
  - Si vous spécifiez une heure, cette valeur sera convertie en format 24 heures. Par exemple, si la date actuelle est le premier janvier 2003, `TS('T16 :04')` ou `TS('16 :04')` est identique à `TS('2003-01-01T16 :04 :00')`.

Exemples de clauses WHERE

- Comparaison d'un ID d'objet avec un ID existant

  ```
  "WORK_ITEM.WIID =
  ID('_WI :800c00ed.df8d7e7c.feffff80.38')"
  ```

  Ce type de clause WHERE est d'habitude créé de façon dynamique avec un ID d'objet existant, obtenu d'un appel antérieur. Si cet ID d'objet est stocké dans une variable *wiid1*, la clause peut être générée comme :

  ```
  "WORK_ITEM.WIID = ID('" + wiid1.String() +
  "')"
  ```

- Utilisation des horodatages

  ```
  "ACTIVITY.STARTED >= TS('2002-06-1T16.00.00')"
  ```

- Utilisation des constantes symboliques

  ```
  "WORK_ITEM.REASON =
  WORK_ITEM.REASON.REASON_OWNER"
  ```

- Utilisation des valeurs booléennes vrai et faux

  ```
  "ACTIVITY.BUSINESS_RELEVANCE = TRUE"
  ```

- Utilisation de propriétés personnalisées

  ```
  "TASK_CPROP1.NAME = 'prop1' AND " TASK_CPROP1.STRING_VALUE = 'v1' AND
   TASK_CPROP2.NAME = 'prop2' AND " TASK_CPROP2.STRING_VALUE = 'v2'"
  ```

Clause Order-by :

La clause ORDER BY de la fonction de requête spécifie les critères de tri pour l'ensemble de résultats de la requête.

Vous pouvez spécifier la liste des colonnes à partir des vues servant de base de tri du résultat. Ces colonnes doivent être entièrement qualifiées par le nom de la vue et de la colonne.

La syntaxe de la clause Order-by est similaire à la syntaxe d'une clause SQL Order-by. Utilisez une virgule pour séparer chaque partie de la clause. Vous pouvez également spécifier la commande `ASC` pour trier les colonnes dans l'ordre

croissant et la commande DESC pour les trier dans l'ordre décroissant. Si vous ne désirez pas trier l'ensemble de résultats, spécifiez la valeur null pour la clause ORDER BY.

Des critères de tri sont appliqués au serveur ; en fait, ce sont les paramètres régionaux du serveur qui sont utilisés pour le tri. Si la requête spécifie plusieurs propriétés, l'ensemble de résultats est trié par les valeurs de la première colonne et ensuite par les valeurs de la deuxième propriété, et ainsi de suite. Contrairement à la requête SQL, il est impossible de spécifier les colonnes dans la clause Order-by par position.

Exemples de clauses ORDER BY

- "PROCESS_TEMPLATE.NAME"

  Trie les résultats de la requête alphabétiquement par le nom du modèle de processus.
- "PROCESS_INSTANCE.CREATED, PROCESS_INSTANCE.NAME DESC"

  Trie les résultats de la requête par date de création, et pour une date spécifique, trie les résultats alphabétiquement pas le nom de l'instance du processus en ordre inverse.
- "ACTIVITY.OWNER, ACTIVITY.TEMPLATE_NAME, ACTIVITY.STATE"

  Trie les résultats de la requête par le propriétaire de l'activité, ensuite par le nom du modèle d'activité et ensuite par l'état de l'activité.

Paramètre Skip-tuples :

Le paramètre skip-tuples spécifie le nombre de tuples dans l'ensemble de résultats de requête, en partant du début, à ignorer et à ne pas renvoyer à l'appelant dans l'ensemble des résultats de requête.

Utilisez ce paramètre avec le paramètre threshold pour implémenter la pagination dans une application client, par exemple, pour extraire les 20 premiers éléments, puis les 20 éléments suivants, etc.

Si ce paramètre a pour valeur null et que le paramètre threshold n'est pas défini, tous les tuples correspondants sont renvoyés.

Exemple de paramètre skip-tuples

- new Integer(5)

  Spécifie que les cinq premiers tuples correspondants ne seront pas renvoyés.

Paramètre Threshold :

Le paramètre threshold de la fonction de requête restreint le nombre d'objets renvoyés du serveur au client dans l'ensemble de résultats de requête.

Dans un environnement de production, les ensembles de résultats d'une requête peuvent contenir des milliers voire des millions d'éléments. Pour cette raison, il est recommandé de toujours définir un seuil. Si vous définissez le paramètre threshold correctement, la requête dans la base de données est plus rapide et moins de données sont transférées à partir du serveur vers le client. Le paramètre threshold peut s'avérer utile, par exemple, dans une interface utilisateur graphique où seul un petit nombre d'éléments peuvent être affichés en même temps.

Si ce paramètre a pour valeur null et que le paramètre skip-tuples n'est pas défini, tous les objets correspondants sont renvoyés.

Exemple de paramètre threshold

- new Integer(50)

  Spécifie que 50 tuples correspondants doivent être renvoyés.

Paramètre Timezone :

Le paramètre time-zone de la fonction de requête définit le fuseau horaire des constantes d'horodatage de la requête.

Le fuseau horaire du client qui lance la requête peut différer de celui du serveur qui traite la requête. Utilisez le paramètre time-zone pour spécifier le fuseau horaire des constantes d'horodatage dans la clause WHERE utilisées, par exemple, pour spécifier l'heure locale. Les dates renvoyées dans l'ensemble de résultats de la requête sont dans le fuseau horaire spécifié pour la requête.

Si le paramètre a pour valeur `null`, les valeurs par défaut des constantes d'horodatage sont en temps universel UTC.

Exemples de paramètres time-zone

- ```
  process.query("ACTIVITY.AIID",
                "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
                 (Chaîne)null,
                 (Entier)null,
                 java.util.TimeZone.getDefault() );
  ```

  Renvoie les ID d'objet pour les activités démarrées après 17h40 heure locale, le premier janvier 2005.
- ```
  process.query("ACTIVITY.AIID",
                "ACTIVITY.STARTED > TS('2005-01-01T17:40')",
                 (Chaîne)null, (Entier)null, (FuseauHoraire)null);
  ```

  Renvoie les ID d'objet pour les activités démarrées après 17h40 UTC, le premier janvier 2005. Cette spécification est décalée de 6 heures en heure EST (Eastern Standard Time).

Filtrage de données à l'aide de variables définies dans des requêtes :

Un résultat de requête renvoie l'objet répondant aux critères de la recherche. Vous pouvez filtrer ces résultats selon les valeurs des variables.

task_context

Vous pouvez définir des variables utilisées par un processus lors de l'exécution dans son modèle de processus. Vous pouvez, pour ces variables, déclarer sur quelles parties porte la requête.

Voici un exemple : John Smith appelle sa société d'assurance afin de connaître le statut de sa demande d'indemnisation suite à un accident de la circulation. L'administrateur des demandes d'indemnisation recherche le dossier du client par le biais de son ID client.

task_procedure

1. OptionalColonSymbol Répertoriez les propriétés des variables dans un processus pouvant faire l'objet d'une requête.

   Identifiez le processus par le biais de l'ID du modèle de processus. Vous pouvez omettre cette étape si vous connaissez les variables pouvant faire l'objet de requêtes.

```
List variableProperties = process.getQueryProperties(ptid);
for (int i = 0; i < variableProperties.size(); i++)
{
   QueryProperty queryData = (QueryProperty)variableProperties.get(i);
   String variableName = queryData.getVariableName();
   String name         = queryData.getName();
   int mappedType      = queryData.getMappedType();
   ...
}
```

2. Dressez la liste des instances de processus contenant les variables conformes aux critères de filtrage.

   Pour ce processus, l'ID client est modélisé en tant que partie de la variable customerClaim pouvant être soumise à la requête. Ainsi, vous pouvez rechercher la demande d'indemnisation par l'intermédiaire de l'ID client.

   ```
   QueryResultSet result = process.query
       ("PROCESS_INSTANCE.NAME, QUERY_PROPERTY.STRING_VALUE",
        "QUERY_PROPERTY.VARIABLE_NAME = 'customerClaim' AND " +
        "QUERY_PROPERTY.NAME = 'customerID' AND " +
        "QUERY_PROPERTY.STRING_VALUE like 'Smith%'",
         (String)null, (Integer)null,
         (Integer)null, (TimeZone)null );
   ```

   Cette opération renvoie un ensemble de résultats de requête contenant les noms d'instance de processus et les valeurs d'ID des clients dont l'identifiant commence par 'Smith'.

Résultats d'interrogation :

Un ensemble de résultats de requête contient les résultats d'une requête d'API de Business Process Choreographer.

Les éléments de l'ensemble de résultats sont les propriétés des objets qui sont conformes à la clause Where fournie par l'appelant et que ce dernier est autorisé à voir. Vous pouvez lire les éléments d'une manière relative à l'aide de la méthode next de l'API ou d'une manière absolue à l'aide des méthodes first et last. Le curseur implicite d'un ensemble de résultats de requête étant positionné, au départ, avant le premier élément, vous devez appeler la méthode first ou next avant de lire un élément. Vous pouvez utiliser la méthode size pour déterminer le nombre d'éléments d'un ensemble.

Un élément de l'ensemble de résultats de la recherche comprend les attributs sélectionnés des éléments de travail et les objets référencés y associés, tels que les instances d'activité et les instances de processus. Le premier attribut (colonne) d'un élément QueryResultSet spécifie la valeur du premier attribut spécifié dans la clause SELECT de la demande de requête. Le deuxième attribut (colonne) d'un élément QueryResultSet spécifie la valeur du deuxième attribut spécifié dans la clause SELECT de la demande de requête et ainsi de suite.

Vous pouvez extraire les valeurs des attributs en appelant une méthode compatible avec le type d'attribut et en spécifiant l'indice de colonne correspondant. La numérotation des indices de colonnes commence à 1.

Type d'attribut	Méthode
Chaîne	getString
OID	getOID

Type d'attribut	Méthode
Horodatage	getTimestamp getString getTimestampAsLong
Entier	getInteger getShort getLong getString getBoolean
Booléen	getBoolean getShort getInteger getLong getString
byte[]	getBinary

Exemple :

La requête suivante est exécutée :

```
QueryResultSet resultSet = process.query("ACTIVITY.STARTED,
                                         ACTIVITY.TEMPLATE_NAME AS NAME,
                                         WORK_ITEM.WIID, WORK_ITEM.REASON",
                                         (String)null, (String)null,
                                         (Integer)null, (TimeZone)null);
```

L'ensemble de résultats renvoyé a quatre colonnes :
- La colonne 1 est l'horodatage
- La colonne 2 est une chaîne
- La colonne 3 est un ID d'objet
- La colonne 4 est un entier

Les méthodes suivantes vous permettent d'obtenir les valeurs des attributs :

```
while (resultSet.next())
{
 java.util.Calendar activityStarted = resultSet.getTimestamp(1);
 String templateName = resultSet.getString(2);
 WIID wiid = (WIID) resultSet.getOID(3);
 Integer reason = resultSet.getInteger(4);
}
```

Vous pouvez utiliser les noms affichés de l'ensemble de résultats, par exemple, en tant qu'en-têtes d'un tableau imprimé. Ces noms sont les noms de colonnes de la vue ou du nom défini par la clause AS dans la requête. Cet exemple illustre l'utilisation de la méthode suivante pour obtenir les noms affichés :

```
resultSet.getColumnDisplayName(1) returns "STARTED"
resultSet.getColumnDisplayName(2) returns "NAME"
resultSet.getColumnDisplayName(3) returns "WIID"
resultSet.getColumnDisplayName(4) returns "REASON"
```

Conditions d'accès propres à l'utilisateur

Les conditions d'accès propres à l'utilisateur sont ajoutées lorsque l'instruction SQL SELECT est générée par la requête API. Ces conditions garantissent que seuls ces objets sont renvoyés à l'appelant parce que conformes à la condition spécifiée par l'appelant et rendus accessibles à ce dernier.

La condition d'accès n'est ajoutée que si l'utilisateur est un administrateur système.

Requêtes appelées par les utilisateurs autres que les administrateurs système

La clause SQL générée WHERE combine l'API avec la clause dotée d'une condition de contrôle d'accès qui est propre à l'utilisateur. La requête n'extrait que les objets auxquels l'utilisateur est autorisé à accéder, autrement dit, uniquement les objets pour lesquels l'utilisateur dispose d'un élément de travail. Un élément de travail représente l'affectation du rôle d'autorisation d'un objet métier à un utilisateur ou un groupe, comme une tâche ou un processus. Par exemple, si l'utilisateur, John Smith, est un membre doté du rôle de propriétaire potentiel d'une tâche donnée, un objet élément de travail existe pour représenter cette relation.

Par exemple, si un utilisateur autre qu'un administrateur système, requiert des tâches, la condition d'accès suivante est ajoutée à la clause WHERE si les éléments de travail de groupe ne sont pas activés :

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'user'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

De ce fait, si John Smith souhaite obtenir la liste des tâches dont il est propriétaire potentiel, l'API contenant la clause se présentera comme suit :

```
"WORK_ITEM.REASON == WORK_ITEM.REASON.REASON_POTENTIAL_OWNER"
```

Cette API contenant la clause génère la condition d'accès suivante dans l'instruction SQL :

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
AND ( WI.OWNER_ID = 'JohnSmith'
      OR WI.OWNER_ID = null AND WI.EVERYBODY = true)
AND WI.REASON = 1
```

Cela signifie également que si John Smith souhaite voir les activités et les tâches dont il est lecteur de processus ou administrateur de processus et pour lesquelles il dispose d'un élément de travail, une propriété provenant de la vue PROCESS_INSTANCE doit être ajoutée à la clause Select, Where, ou Order-by de la requête, telle que PROCESS_INSTANCE.PIID.

Si les éléments de travail de groupe sont activés, une autre condition d'accès est ajoutée à la clause WHERE qui permet à un utilisateur d'accéder aux objets auxquels le groupe a accès.

Requêtes appelées par les administrateurs système

Les administrateurs système peuvent appeler la méthode query pour extraire des objets dotés d'éléments de travail associés. Dans ce cas, un joint à la vue WORK_ITEM est ajouté à la requête SQL générée, mais sans condition de contrôle d'accès pour WORK_ITEM.OWNER_ID.

Dans ce cas, la requête SQL des tâches contient ce qui suit :

```
FROM TASK TA, WORK_ITEM WI
WHERE WI.OBJECT_ID = TA.TKIID
```

queryAll queries

Ce type de requête ne peut être appelé que par les administrateurs système ou les contrôleurs système. Ni les conditions de contrôle d'accès, ni un joint à la vue

WORK_ITEM ne sont ajoutés. Ce type de requête renvoie toutes les données de tous les objets.

Exemples de méthodes query et queryAll

Ces exemples montrent la syntaxe de diverses requêtes API générales et des instructions SQL associées qui sont générées lors du traitement de la requête.

Concepts associés

«Exemple : requête de tâches à l'état Prêt»
Cet exemple indique comment utiliser la méthode query pour extraire les tâches que l'utilisateur connecté peut exploiter.

«Exemple : requête de tâches à l'état Réclamé», à la page 430
Cet exemple indique comment utiliser la méthode query pour extraire les tâches que l'utilisateur connecté a réclamées.

«Exemple : interrogation d'escalades», à la page 431
Cet exemple indique comment utiliser la méthode query pour extraire les escalades pour l'utilisateur connecté.

«Exemple : utilisation de la méthode queryAll», à la page 431
Cet exemple indique comment utiliser la méthode queryAll pour extraire toutes les activités propres à un modèle de processus.

«Exemple : ajout de propriétés de requête à une requête», à la page 432
Cet exemple indique comment utiliser la méthode query pour extraire les tâches propres à un processus métier. Le processus dispose de propriétés de requête spécifiques que vous pouvez inclure à la recherche.

«Exemple : ajout de propriétés personnalisées à une requête», à la page 433
Cet exemple montre comment utiliser la méthode query pour extraire les tâches dotées de propriétés personnalisées.

Exemple : requête de tâches à l'état Prêt :

Cet exemple indique comment utiliser la méthode query pour extraire les tâches que l'utilisateur connecté peut exploiter.

John Smith souhaite obtenir la liste des tâches qui lui ont été affectées. Pour qu'un utilisateur puisse travailler sur une tâche, celle-ci doit avoir l'état Prêt. L'utilisateur connecté doit également avoir l'élément de travail d'un propriétaire potentiel de la tâche. Le fragment de code suivant affiche l'appel de méthode query pour cette requête :

```
query( "DISTINCT TASK.TKIID",
           "TASK.KIND IN ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )
             AND " +
           "TASK.STATE = TASK.STATE.STATE_READY AND " +
           "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
            (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Les actions suivantes sont prises lorsque l'instruction SQL SELECT est générée :

- Une condition pour le contrôle d'accès est ajoutée à la clause Where. Cet exemple suppose que les éléments de travail de ce groupe ne sont pas activés.
- Les constantes, telles que `TASK.STATE.STATE_READY` sont remplacées par leurs valeurs numériques.
- Une clause FROM et les conditions de joint sont ajoutées.

Le fragment de code suivant montre l'instruction SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.KIND IN ( 101, 105 )
   AND    TA.STATE = 2
   AND    WI.REASON = 1
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND  WI.EVERYBODY = true )
```

Pour restreindre la requête API aux tâches d'un processus spécifique, par exemple, sampleProcess, la requête ressemble à ce qui suit :

```
query( "DISTINCT TASK.TKIID",
           "PROCESS_TEMPLATE.NAME = 'sampleProcess' AND "+
           "TASK.KIND IN ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING )
             AND " +
           "TASK.STATE = TASK.STATE.STATE_READY AND " +
           "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
            (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Exemple : requête de tâches à l'état Réclamé :

Cet exemple indique comment utiliser la méthode query pour extraire les tâches que l'utilisateur connecté a réclamées.

L'utilisateur, John Smith, souhaite rechercher des tâches qu'il a réclamées et qui sont toujours à l'état Réclamé. La condition qui spécifie "réclamé par John Smith" est `TASK.OWNER = 'JohnSmith'`. Le fragment de code suivant indique l'appel de méthode query pour la requête :

```
query( "DISTINCT TASK.TKIID",
           "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +
           "TASK.OWNER = 'JohnSmith'",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Le fragment de code suivant montre l'instruction SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.STATE = 8
   TA.OWNER = 'JohnSmith'
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

Lorsqu'une tâche est réclamée, les éléments de travail sont créés pour le propriétaire de la tâche. Ainsi, l'autre façon de former la requête pour les tâches réclamées de John Smith consiste à ajouter la condition suivante à la requête au lieu d'utiliser `TASK.OWNER = 'JohnSmith'`:

```
WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER
```

Alors la requête ressemble au fragment de code suivant :

```
query( "DISTINCT TASK.TKIID",
           "TASK.STATE = TASK.STATE.STATE_CLAIMED AND " +
           "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_OWNER",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Les actions suivantes sont prises lorsque l'instruction SQL SELECT est générée :

- Une condition pour le contrôle d'accès est ajoutée à la clause Where. Cet exemple suppose que les éléments de travail de ce groupe ne sont pas activés.
- Les constantes, telles que `TASK.STATE.STATE_READY` sont remplacées par leurs valeurs numériques.
- Une clause FROM et les conditions de joint sont ajoutées.

Le fragment de code suivant montre l'instruction SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  WI.OBJECT_ID = TA.TKIID
   AND    TA.STATE = 8
   AND    WI.REASON = 4
   AND  ( WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID = null AND WI.EVERYBODY = true )
```

John est sur le point de partir en congés, donc son responsable d'équipe, Anne Grant, souhaite évaluer sa charge de travail actuelle. Anne dispose des droits d'administration. La requête qu'elle appelle est la même que celle appelée par John. Cependant, l'instruction SQL qui est générée est différente car Anne est administrateur. Le fragment de code suivant indique l'instruction SQL générée :

```
SELECT DISTINCT TASK.TKIID
   FROM   TASK TA, WORK_ITEM WI,
   WHERE  TA.TKIID = WI.OBJECT_ID =
   AND    TA.STATE = 8
   AND   TA.OWNER = 'JohnSmith')
```

Du fait qu'Anne est administrateur, une condition de contrôle d'accès n'est pas ajoutée à la clause WHERE.

Exemple : interrogation d'escalades :

Cet exemple indique comment utiliser la méthode query pour extraire les escalades pour l'utilisateur connecté.

Lorsqu'une tâche est escaladée, un élément de travail récepteur d'escalade est créé. L'utilisateur Mary Jones souhaite voir la liste des tâches qui lui ont été escaladées. Le fragment de code suivant indique l'appel de méthode query pour la requête :

```
query( "DISTINCT ESCALATION.ESIID, ESCALATION.TKIID",
         "WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_ESCALATION_RECEIVER",
           (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Les actions suivantes sont prises lorsque l'instruction SQL SELECT est générée :

- Une condition pour le contrôle d'accès est ajoutée à la clause Where. Cet exemple suppose que les éléments de travail de ce groupe ne sont pas activés.
- Les constantes, telles que `TASK.STATE.STATE_READY` sont remplacées par leurs valeurs numériques.
- Une clause FROM et les conditions de joint sont ajoutées.

Le fragment de code suivant indique l'instruction SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT ESCALATION.ESIID, ESCALATION.TKIID
  FROM   ESCALATION ESC, WORK_ITEM WI
  WHERE  ESC.ESIID = WI.OBJECT_ID
  AND    WI.REASON  = 10
  AND
 ( WI.OWNER_ID = 'MaryJones' OR WI.OWNER_ID = null AND  WI.EVERYBODY = true )
```

Exemple : utilisation de la méthode queryAll :

Cet exemple indique comment utiliser la méthode queryAll pour extraire toutes les activités propres à un modèle de processus.

La méthode queryAll est disponible uniquement pour les utilisateurs avec des droits d'administrateur système ou de contrôleur système. Le fragment de code suivant indique l'appel de méthode queryAll pour la requête permettant d'extraire toutes les activités propres au modèle de processus, sampleProcess :

```
queryAll( "DISTINCT ACTIVITY.AIID",
       "PROCESS_TEMPLATE.NAME = 'sampleProcess'",
       (String)null, (String)null, (Integer)null, (TimeZone)null )
```

Le fragment de code suivant montre la requête SQL qui est générée à partir de la requête API :

```
SELECT DISTINCT ACTIVITY.AIID
  FROM   ACTIVITY AI, PROCESS_TEMPLATE PT
  WHERE  AI.PTID = PT.PTID
  AND    PT.NAME = 'sampleProcess'
```

Du fait que l'appel est invoqué par un administrateur, une condition de contrôle d'accès n'est pas ajoutée à l'instruction SQL générée. Un joint à la vue WORK_ITEM n'est pas ajouté non plus. Cela signifie que la requête extrait toutes les activités du modèle de processus, y compris les activités sans élément de travail.

Exemple : ajout de propriétés de requête à une requête :

Cet exemple indique comment utiliser la méthode query pour extraire les tâches propres à un processus métier. Le processus dispose de propriétés de requête spécifiques que vous pouvez inclure à la recherche.

Par exemple, vous souhaitez rechercher toutes les tâches manuelles à l'état Prêt qui sont propres à un processus métier. Le processus fournit la propriété de requête **customerID** qui est dotée de la valeur CID_12345 et d'un espace de nom. Le fragment de code suivant indique l'appel de méthode query pour la requête :

```
query ( " DISTINCT TASK.TKIID, TASK_TEMPL.NAME, TASK.STATE,
                  PROCESS_INSTANCE.NAME",
          " QUERY_PROPERTY.NAME = 'customerID' AND " +
          " QUERY_PROPERTY.STRING_VALUE = 'CID_12345' AND " +
          " QUERY_PROPERTY.NAMESPACE =
             'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

A présent, si vous souhaitez ajouter une deuxième propriété de requête à la requête, comme par exemple, **Priority**, avec un espace de nom donné, l'appel de méthode query de la requête ressemble à ce qui suit :

```
query ( " DISTINCT TASK.TKIID, TASK_TEMPL.NAME, TASK.STATE,
                  PROCESS_INSTANCE.NAME",
          " QUERY_PROPERTY1.NAME = 'customerID' AND " +
          " QUERY_PROPERTY1.STRING_VALUE = 'CID_12345' AND " +
          " QUERY_PROPERTY1.NAMESPACE =
                 'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
          " QUERY_PROPERTY2.NAME = 'Priority' AND " +
          " QUERY_PROPERTY2.NAMESPACE =
                'http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/' AND " +
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

Si vous ajoutez plusieurs propriétés de requête à la requête, vous devez numéroter chaque propriété que vous ajoutez comme indiqué dans le fragment de code. Cependant, l'interrogation des propriétés personnalisées a une répercussion sur les performances, car elles se réduisent du fait du nombre de propriétés personnalisées dans la requête.

Exemple : ajout de propriétés personnalisées à une requête :

Cet exemple montre comment utiliser la méthode query pour extraire les tâches dotées de propriétés personnalisées.

Par exemple, vous souhaitez rechercher toutes les tâches manuelles à l'état Prêt avec la propriété personnalisée **customerID** et la valeur `CID_12345`. Le fragment de code suivant indique l'appel de méthode query pour la requête :

```
query( "DISTINCT TASK.TKIID",
          " TASK_CPROP.NAME = 'customerID' AND " +
          " TASK_CPROP.STRING_VALUE = 'CID_12345' AND " +
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

A présent, si vous souhaitez extraire les tâches et leurs propriétés personnalisées, l'appel de méthode query de la requête ressemble à ce qui suit :

```
query ( " DISTINCT TASK.TKIID, TASK_CPROP.NAME, TASK_CPROP.STRING_VALUE",
          " TASK.KIND IN
             ( TASK.KIND.KIND_HUMAN, TASK.KIND.KIND_PARTICIPATING ) AND " +
          " TASK.STATE = TASK.STATE.STATE_READY ",
          (String)null, (String)null, (Integer)null, (TimeZone)null );
```

L'instruction SQL qui est générée à partir de cette requête API s'affiche dans le fragment de code suivant :

```
SELECT DISTINCT TA.TKIID , TACP.NAME , TACP.STRING_VALUE
   FROM  TASK TA LEFT JOIN TASK_CPROP TACP ON (TA.TKIID = TACP.TKIID),
         WORK_ITEM WI
   WHERE WI.OBJECT_ID = TA.TKIID
   AND    TA.KIND IN ( 101, 105 )
   AND    TA.STATE = 2
   AND  (WI.OWNER_ID = 'JohnSmith' OR WI.OWNER_ID IS NULL AND WI.EVERYBODY = 1 )
```

Cette instruction SQL contient un joint extérieur entre la vue TASK et la vue TASK_CPROP. Cela signifie que les tâches qui répondent à la clause WHERE sont extraites même si elles ne comportent pas de propriété personnalisée.

Gestion des requêtes stockées

Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

task_context

Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Une requête privée et une requête publique peuvent être sauvegardées sous le même nom. Les requêtes enregistrées par différents utilisateurs peuvent également avoir un nom identique.

Vous pouvez avoir stocké des requêtes pour des objets de processus métier, des objets de tâche ou une combinaison de ces deux types d'objets.

Concepts associés

«Paramètres des requêtes stockées»
Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Les uplets répondant aux critères sont assemblés de manière dynamique lors de l'exécution de la requête. Pour rendre les requêtes stockées réutilisables, vous pouvez utiliser les paramètres de la définition de requête résolus lors de l'exécution.

«Paramètres des requêtes stockées»
Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Les uplets répondant aux critères sont assemblés de manière dynamique lors de l'exécution de la requête. Pour rendre les requêtes stockées réutilisables, vous pouvez utiliser les paramètres de la définition de requête résolus lors de l'exécution.

Tâches associées

«Gestion des requêtes stockées publiques», à la page 435
Les requêtes stockées publiques sont créées par l'administrateur système. Ces requêtes sont accessibles à tous les utilisateurs.

«Gestion des requêtes stockées privées d'autres utilisateurs», à la page 436
Tout utilisateur peut créer des requêtes privées. Seul le propriétaire d'une requête et l'administrateur système peuvent les utiliser.

«Gestion des requêtes stockées privées», à la page 437
Si vous n'êtes pas un administrateur système, vous pouvez créer, exécuter et supprimer vos propres requêtes stockées privées. Vous pouvez également utiliser les requêtes stockées publiques créées par l'administrateur système.

Paramètres des requêtes stockées :

Une requête stockée est une requête qui est enregistrée dans la base de données et identifiée par un nom. Les uplets répondant aux critères sont assemblés de manière dynamique lors de l'exécution de la requête. Pour rendre les requêtes stockées réutilisables, vous pouvez utiliser les paramètres de la définition de requête résolus lors de l'exécution.

Il existe par exemple des propriétés personnalisées pour stocker les noms de client. Vous pouvez définir des requêtes visant à renvoyer les tâches associées à un client donné, ACME Co. Pour faire la demande de ces informations, la clause where de votre requête devrait ressembler à ce qui est indiqué dans l'exemple suivant :

```
String whereClause =
  "TASK.STATE = TASK.STATE.STATE_READY
  AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER
  AND TASK_CPROP.NAME = 'company' AND TASK_CPROP.STRING_VALUE = 'ACME Co.'";
```

Pour rendre cette requête réutilisable afin de permettre également la recherche du client BCME Ltd, vous pouvez configurer des paramètres pour les valeurs de la propriété personnalisée. Si vous ajoutez des paramètres à la requête, celle-ci se peut présenter comme suit :

```
String whereClause =
  "TASK.STATE = TASK.STATE.STATE_READY
  AND WORK_ITEM.REASON = WORK_ITEM.REASON.REASON_POTENTIAL_OWNER
  AND TASK_CPROP.NAME = 'company' AND TASK_CPROP.STRING_VALUE = '@param1'";
```

Le paramètre `@param1` est résolu au moment de l'exécution à partir de la liste des paramètres transmis à la méthode query. Les règles suivantes s'appliquent lors de l'utilisation de paramètres dans les requêtes :

- Les paramètres sont utilisables uniquement dans la clause where.
- Les paramètres sont de type Chaîne.
- Les paramètres sont remplacés au moment de l'exécution via une substitution de chaînes. Si des caractères spéciaux sont nécessaires, vous devez les spécifier dans la clause where ou les insérer au moment de l'exécution en tant que partie du paramètre.
- Les noms de paramètre sont constitués de la chaîne `@param` concaténée avec un nombre entier. La valeur la plus faible est 1, ce qui renvoie au premier élément de la liste des paramètres transmis à l'API de la requête au moment de l'exécution.
- Un paramètre peut être réutilisé plusieurs fois au sein d'une clause where ; toutes les occurrences du paramètre sont remplacées par la même valeur.

Tâches associées

«Gestion des requêtes stockées», à la page 433
Les requêtes stockées permettent d'enregistrer des requêtes souvent exécutées. La requête stockée peut soit être une requête disponible pour tous les utilisateurs (requête publique), soit une requête appartenant à un utilisateur spécifique (requête privée).

Gestion des requêtes stockées publiques :

Les requêtes stockées publiques sont créées par l'administrateur système. Ces requêtes sont accessibles à tous les utilisateurs.

task_context

En tant qu'administrateur système, vous pouvez créer, visualiser et supprimer des requêtes stockées publiques. Si vous ne spécifiez aucun ID utilisateur dans l'appel d'API, on suppose que la requête stockée est une requête stockée publique.

task_procedure

1. Créez une requête stockée publique.

   Par exemple, le fragment de code suivant crée une requête stockée pour les instances de processus et l'enregistre sous le nom CustomerOrdersStartingWithA.

   ```
   process.createStoredQuery("CustomerOrdersStartingWithA",
                "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
                "PROCESS_INSTANCE.NAME LIKE 'A%'",
                "PROCESS_INSTANCE.NAME",
                 (Integer)null, (TimeZone)null);
   ```

   Le résultat de la requête stockée consiste en une liste triée de tous les noms d'instance de processus commençant par la lettre A et de leurs identifiants d'instance de processus associés (PIID).
2. Exécutez la requête définie par la requête stockée.

   ```
   QueryResultSet result = process.query("CustomerOrdersStartingWithA",
                  new Integer(0), null);
   ```

   Cette action renvoie les objets qui répondent aux critères. Dans le cas présent, toutes les commandes client commençant par A.
3. Répertoriez les requêtes stockées publiques disponibles.

Le fragment de code suivant vous permet de restreindre aux requêtes publiques la liste des requêtes renvoyées.

```
String[] storedQuery = process.getStoredQueryNames(StoredQueryData.KIND_PUBLIC);
```

4. OptionalColonSymbol Vérifiez la requête définie par une requête stockée spécifique.

   Une requête stockée privée peut porter le même nom qu'une requête stockée publique. Si ces noms sont identiques, la requête stockée renvoyée est la requête privée. Le fragment de code suivant montre comment renvoyer la requête publique portant le nom spécifié. Si vous utilisez l'API de Human Task Manager pour extraire des informations sur une requête stockée, utilisez StoredQuery au lieu de StoredQueryData pour l'objet renvoyé.

```
StoredQueryData storedQuery = process.getStoredQuery
   (StoredQueryData.KIND_PUBLIC, "CustomerOrdersStartingWithA");
String selectClause = storedQuery.getSelectClause();
String whereClause = storedQuery.getWhereClause();
String orderByClause = storedQuery.getOrderByClause();
Integer threshold = storedQuery.getThreshold();
String owner = storedQuery.getOwner();
```

5. Supprimez une requête stockée publique.

   Le fragment de code suivant montre comment supprimer la requête stockée que vous avez créée à l'étape 1.

```
process.deleteStoredQuery("CustomerOrdersStartingWithA");
```

Gestion des requêtes stockées privées d'autres utilisateurs :

Tout utilisateur peut créer des requêtes privées. Seul le propriétaire d'une requête et l'administrateur système peuvent les utiliser.

task_context

En tant qu'administrateur système, vous pouvez gérer des requêtes stockées privées qui appartiennent à un utilisateur spécifique.

task_procedure

1. Créez une requête stockée privée pour l'ID utilisateur Smith.

   Par exemple, le fragment de code suivant crée une requête stockée pour les instances de processus et l'enregistre sous le nom CustomerOrdersStartingWithA pour l'ID utilisateur Smith.

```
process.createStoredQuery("Smith", "CustomerOrdersStartingWithA",
             "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
             "PROCESS_INSTANCE.NAME LIKE 'A%'",
             "PROCESS_INSTANCE.NAME",
              (Integer)null, (TimeZone)null,
              (List)null, (String)null);
```

   La requête stockée renvoie une liste triée de tous les noms d'instance de processus commençant par la lettre A et de leurs identifiants d'instance de processus associés (PIID).

2. Exécutez la requête définie par la requête stockée.

```
QueryResultSet result = process.query
                       ("Smith", "CustomerOrdersStartingWithA",
                         (Integer)null, (Integer)null, (List)null);
new Integer(0));
```

   Cette action renvoie les objets qui répondent aux critères. Dans le cas présent, toutes les commandes client commençant par A.

3. Accédez à la liste des noms des requêtes privées appartenant à un utilisateur donné.

Par exemple, le fragment de code suivant montre comment obtenir la liste des requêtes privées appartenant à l'utilisateur Smith.

```
String[] storedQuery = process.getStoredQueryNames("Smith");
```

4. Affichez les détails d'une requête spécifique.

   Le fragment de code suivant montre comment afficher les détails de la requête CustomerOrdersStartingWithA qui appartient à l'utilisateur Smith.

```
StoredQueryData storedQuery = process.getStoredQuery
   ("Smith", "CustomerOrdersStartingWithA");
String selectClause = storedQuery.getSelectClause();
String whereClause = storedQuery.getWhereClause();
String orderByClause = storedQuery.getOrderByClause();
Integer threshold = storedQuery.getThreshold();
String owner = storedQuery.getOwner();
```

   Si vous utilisez l'API de Human Task Manager pour extraire des informations sur une requête stockée, utilisez StoredQuery au lieu de StoredQueryData pour l'objet renvoyé.

5. Supprimez une requête stockée privée.

   Le fragment de code suivant montre comment supprimer une requête privée qui appartient à l'utilisateur Smith.

```
process.deleteStoredQuery("Smith", "CustomerOrdersStartingWithA");
```

Gestion des requêtes stockées privées :

Si vous n'êtes pas un administrateur système, vous pouvez créer, exécuter et supprimer vos propres requêtes stockées privées. Vous pouvez également utiliser les requêtes stockées publiques créées par l'administrateur système.

task_procedure

1. Créez une requête stockée privée.

   Par exemple, le fragment de code suivant crée une requête stockée pour les instances de processus et l'enregistre sous un nom spécifique. Si aucun ID utilisateur n'est spécifié, on suppose que la requête stockée est une requête stockée privée de l'utilisateur connecté.

```
process.createStoredQuery("CustomerOrdersStartingWithA",
             "DISTINCT PROCESS_INSTANCE.PIID, PROCESS_INSTANCE.NAME",
             "PROCESS_INSTANCE.NAME LIKE 'A%'",
             "PROCESS_INSTANCE.NAME",
              (Integer)null, (TimeZone)null);
```

   Cette requête renvoie une liste triée de tous les noms d'instance de processus commençant par la lettre A et de leurs identifiants d'instance de processus associés (PIID).

2. Exécutez la requête définie par la requête stockée.

```
QueryResultSet result = process.query("CustomerOrdersStartingWithA",
               new Integer(0));
```

   Cette action renvoie les objets qui répondent aux critères. Dans le cas présent, toutes les commandes client commençant par A.

3. Extrayez une liste des noms de requêtes stockées auxquelles l'utilisateur connecté peut accéder.

   Le fragment de code suivant montre comment extraire les requêtes stockées auxquelles l'utilisateur connecté peut accéder.

```
String[] storedQuery = process.getStoredQueryNames();
```

4. Affichez les détails d'une requête spécifique.

Le fragment de code suivant montre comment afficher les détails de la requête CustomerOrdersStartingWithA dont l'utilisateur Smith est le propriétaire.

```
StoredQueryData storedQuery = process.getStoredQuery
   ("CustomerOrdersStartingWithA");
String selectClause = storedQuery.getSelectClause();
String whereClause = storedQuery.getWhereClause();
String orderByClause = storedQuery.getOrderByClause();
Integer threshold = storedQuery.getThreshold();
String owner = storedQuery.getOwner();
```

Si vous utilisez l'API de Human Task Manager pour extraire des informations sur une requête stockée, utilisez StoredQuery au lieu de StoredQueryData pour l'objet renvoyé.

5. Supprimez une requête stockée privée.

   Le fragment de code suivant indique comment supprimer une requête stockée privée.

   ```
   process.deleteStoredQuery("CustomerOrdersStartingWithA");
   ```

Développement d'applications client EJB pour des processus métier et des tâches manuelles

Les API EJB fournissent un ensemble de méthodes génériques pour le développement d'applications client EJB permettant d'utiliser des processus métier et des tâches manuelles installées sur WebSphere Process Server.

task_context

Ces API EJB (Enterprise JavaBeans) permettent de créer des applications client pour effectuer les opérations suivantes :

- Gérer le cycle de vie des processus et des tâches, depuis leur lancement jusqu'à leur suppression finale
- Réparer des activités et des processus
- Gérer et distribuer la charge de travail entre les membres d'un groupe de travail

Les API EJB sont fournies sous forme de deux beans enterprise session sans état :

- L'interface BusinessFlowManagerService fournit les méthodes pour les applications de processus métier.
- L'interface HumanTaskManagerService fournit les méthodes pour les applications basées sur des tâches.

Pour plus d'informations concernant les API EJB, voir la documentation Java dans le package com.ibm.bpe.api et le package com.ibm.task.api.

La procédure suivante offre un aperçu des actions à entreprendre pour développer une application client EJB.

task_procedure

1. Déterminez les fonctionnalités que l'application doit offrir.
2. Décidez quels beans session vous souhaitez utiliser.

   En fonction des scénarios que vous souhaitez implémenter à l'aide de votre application, vous pouvez choisir l'un des beans session ou les deux.
3. Déterminez quels sont les droits requis par les utilisateurs de l'application.

   Les utilisateurs de votre application doivent disposer des rôles d'autorisation appropriés pour pouvoir appeler les méthodes que vous incluez dans celle-ci et

pour visualiser les objets et les attributs des objets renvoyés par ces méthodes. Si une instance du bean session approprié est créée, WebSphere Application Server associe un contexte à cette instance. Le contexte contient des informations relatives à l'ID principal de l'appelant, à la liste d'appartenance au groupe et aux rôles. Ces informations sont utilisées à la vérification des droits d'accès de l'appelant pour chaque appel.

Les informations d'autorisation relatives à chacune des méthodes sont décrites dans Javadoc.

4. Déterminez de quelle façon rendre l'application.

   Les interfaces API EJB peuvent être appelées à distance ou localement.

5. Développez l'application.
   a. Accédez à l'API EJB.
   b. Utilisez l'API EJB pour interagir avec les processus ou les tâches.
      - Recherchez les données.
      - Utilisez les données.

Concepts associés

Mode d'autorisation pour l'administration alternative

Tâches associées

«Accès aux API EJB»
Les API EJB (Enterprise JavaBeans) sont fournies sous forme de deux beans enterprise session sans état. Les applications de processus métier et les applications de tâche accèdent au bean enterprise de session approprié via l'interface home du bean.

«Développement d'applications pour les processus métier», à la page 446
Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

«Développement d'applications pour des tâches manuelles», à la page 477
Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches manuelles sont fournis.

«Développement d'applications pour les processus métier et les tâches manuelles», à la page 496
La plupart des scénarios de processus métier nécessitent la participation de personnes. Par exemple, un processus métier nécessite une interaction manuelle lorsque le processus est démarré ou géré ou lorsque des activités manuelles sont effectuées. Pour supporter de tels scénarios, vous devez utiliser à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

«Gestion des exceptions et des erreurs», à la page 503
Un processus BPEL peut rencontrer une erreur à différents points du processus.

Référence associée

«Interface BusinessFlowManagerService», à la page 473
L'interface BusinessFlowManagerService permet l'accès aux fonctions de processus métier pouvant être appelées par une application client.

«Interface HumanTaskManagerService», à la page 494
L'interface HumanTaskManagerService permet l'accès aux fonctions relatives aux tâches pouvant être appelées par des clients locaux ou distants.

Accès aux API EJB

Les API EJB (Enterprise JavaBeans) sont fournies sous forme de deux beans enterprise session sans état. Les applications de processus métier et les applications de tâche accèdent au bean enterprise de session approprié via l'interface home du bean.

task_context

L'interface BusinessFlowManagerService fournit les méthodes pour les applications de processus métier et l'interface HumanTaskManagerService fournit les méthodes pour les applications basées sur des tâches. Il peut s'agir de n'importe quelle application Java, y compris une autre application Enterprise JavaBeans (EJB).

Tâches associées

«Accès à l'interface distante du bean session»
Une application client EJB de processus métier ou de tâches manuelles accède à l'interface distante du bean session par le biais de l'interface home distante du bean.

«Accès à l'interface locale du bean session», à la page 444
Une application client EJB de processus métier ou de tâches manuelles accède à l'interface locale du bean session par le biais de l'interface home locale du bean.

Accès à l'interface distante du bean session

Une application client EJB de processus métier ou de tâches manuelles accède à l'interface distante du bean session par le biais de l'interface home distante du bean.

task_context

Le bean session peut être soit le bean session BusinessFlowManager pour les applications de processus, soit le bean session HumanTaskManager pour les applications de tâche.

task_procedure

1. Ajoutez à l'interface distante du bean session une référence pointant vers le descripteur de déploiement d'applications. Ajoutez la référence à l'un des fichiers suivants :
   - Le fichier `application-client.xml` pour une application client Java Platform, Enterprise Edition (Java EE)
   - Le fichier `web.xml` pour une application Web
   - Le fichier `ejb-jar.xml` pour une application Enterprise JavaBeans (EJB)

   La référence à l'interface home distante des applications de processus est illustrée dans l'exemple suivant :

   ```
   <ejb-ref>
    <ejb-ref-name>ejb/BusinessFlowManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
    <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
   </ejb-ref>
   ```

   La référence à l'interface home locale des applications de tâche est illustrée dans l'exemple suivant :

   ```
   <ejb-ref>
    <ejb-ref-name>ejb/HumanTaskManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.ibm.task.api.HumanTaskManagerHome</home>
    <remote>com.ibm.task.api.HumanTaskManager</remote>
   </ejb-ref>
   ```

   Si vous utilisez WebSphere Integration Developer pour ajouter la référence EJB au descripteur de déploiement, la liaison de la référence EJB est automatiquement créée lors du déploiement de l'application. Pour plus d'informations concernant l'ajout de références EJB, consultez la documentation WebSphere Integration Developer.
2. Décidez de la méthode adoptée pour fournir les définitions des objets métier.

   Pour utiliser des objets métier dans une application client distante, vous devez avoir accès aux schémas correspondants pour les objets métier (fichiers XSD ou WSDL) utilisés pour l'interaction avec un processus ou une tâche. L'accès à ces fichiers est possible de l'une des manières suivantes :

- Si l'application client n'est pas exécutée dans un environnement géré Java EE, incluez les fichiers dans le fichier EAR de l'application client.
- Si l'application client est une application Web ou un client EJB exécuté dans un environnement géré Java EE, vous pouvez inclure les fichiers dans le fichier EAR de l'application client, soit bénéficier du chargement des artefacts distants.
  a. Utilisez l'interface API EJB createMessage de Business Process Choreographer et les méthodes ClientObjectWrapper.getObject pour charger les définitions d'objet métier distantes de l'application correspondante vers le serveur, de façon transparente.
  b. Utilisez l'interface de programmation Service Data Object pour créer ou consulter un objet métier en tant que partie d'un objet métier déjà instancié. Pour cela, utilisez les méthodes commonj.sdo.DataObject.createDataObject ou getDataObject sur l'interface DataObject.
  c. Lorsque vous souhaitez créer un objet métier en tant que valeur de propriété d'un objet métier saisie à l'aide du schéma XML any ou anyType, utilisez les services Business Object pour créer ou lire votre objet métier. Pour cela, vous devez définir le contexte de RAL de manière à pointer vers l'application à partir de laquelle les schémas seront chargés. Vous pouvez ensuite les services des objets métier appropriés.

     Créez par exemple un objet métier dans lequel "ApplicationName" est le nom de l'application qui contient les définitions de vos objets métier.

```
BOFactory bofactory = (BOFactory) new
   ServiceManager().locateService("com/ibm/websphere/bo/BOFactory");

com.ibm.wsspi.al.ALContext.setContext
   ("RALTemplateName", "ApplicationName");
try {
     DataObject dataObject = bofactory.create("uriName", "typeName" );
   } finally {
     com.ibm.wsspi.al.ALContext.unset();
   }
```

     Lisez par exemple une entrée XML dans laquelle "ApplicationName" est le nom de l'application qui contient les définitions de vos objets métier.

```
BOXMLSerializer serializerService =
  (BOXMLSerializer) new ServiceManager().locateService
    ("com/ibm/websphere/bo/BOXMLSerializer");
ByteArrayInputStream input = new ByteArrayInputStream(<?xml?>..);

com.ibm.wsspi.al.ALContext.setContext
          ("RALTemplateName", "ApplicationName");
try {
     BOXMLDocument document = serializerService.readXMLDocument(input);
     DataObject dataObject = document.getDataObject();
   } finally {
     com.ibm.wsspi.al.ALContext.unset();
   }
```

3. Localisez l'interface home distante du bean session dans l'interface JNDI (Java Naming and Directory Interface).

   L'exemple suivant illustre cette étape pour une application de processus :

```
// Obtenir le contexte JNDI initial par défaut
InitialContext initialContext = new InitialContext();

// Rechercher l'interface home distante du bean BusinessFlowManager
Object result =
       initialContext.lookup("java  :comp/env/ejb/BusinessFlowManagerHome");
```

```
// Convertir le résultat de la recherche dans le type approprié
   BusinessFlowManagerHome processHome =
           (BusinessFlowManagerHome)javax.rmi.PortableRemoteObject.narrow
           (result,BusinessFlowManagerHome.class);
```

L'interface home distante du bean session contient une méthode de création pour les objets EJB. Cette méthode renvoie l'interface distante du bean session.

4. Accédez à l'interface distante du bean session.

   L'exemple suivant illustre cette étape pour une application de processus :

   ```
   BusinessFlowManager process = processHome.create();
   ```

   L'accès au bean session ne garantit pas que l'appelant puisse effectuer toutes les actions sur un certain processus ; l'appelant doit être également autorisé à effectuer l'action. Lorsqu'une instance du bean session est créée, elle est associée à un contexte du bean session. Le contexte contient l'ID principal de l'appelant, la liste d'appartenance au groupe, et indique si l'appelant est titulaire d'un des rôles Java EE Business Process Choreographer. Le contexte est utilisé pour vérifier l'autorisation de l'appelant pour chaque appel, même lorsque la sécurité administrative n'est pas configurée. Si la sécurité administrative n'est pas configurée, la valeur de l'ID principal de l'appelant est `UNAUTHENTICATED`.

5. Appelez les fonctions métier exposées par l'interface de service.

   L'exemple suivant illustre cette étape pour une application de processus :

   ```
   process.initiate("MyProcessModel",input);
   ```

   Les appels venant des applications sont exécutés comme des transactions. Une transaction est établie et terminée de l'une des façons suivantes :

   - Automatiquement par WebSphere Application Server (le descripteur de déploiement spécifie TX_REQUIRED).
   - De manière explicite par l'application. Vous pouvez regrouper les appels d'application à l'intérieur d'une seule transaction :

     ```
     // Obtenir l'interface de transaction utilisateur
     UserTransaction transaction=
            (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

     // Commencer une transaction
     transaction.begin();

     // Appels d'applications ...

     // En cas d'aboutissement, valider la transaction
     transaction.commit();
     ```

   Tip : Afin d'éviter tout conflit de verrouillage de la base de données, évitez d'exécuter en parallèle des instructions similaires à la suivante :

   ```
   // Obtenir l'interface de transaction utilisateur
   UserTransaction transaction=
          (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

   transaction.begin();

   //Lire l'instance d'activité
   process.getActivityInstance(aiid);
   //Réclamer l'instance d'activité
   process.claim(aiid);

   transaction.commit();
   ```

   La méthode getActivityInstance ainsi que d'autres opérations de lecture définissent un verrou en lecture. Dans cet exemple, un verrou en lecture sur

l'instance d'activité est mis à niveau vers un verrou U sur l'instance d'activité. Ceci peut provoquer un blocage de la base de données lorsque ces transactions sont exécutées en parallèle.

Exemple

Voici un exemple illustrant les étapes 3 à 5 pour une application de tâche.

```
// Obtenir le contexte JNDI initial par défaut
InitialContext initialContext = new InitialContext();

// Rechercher l'interface home distante du bean HumanTaskManager
Object result =
         initialContext.lookup("java:comp/env/ejb/HumanTaskManagerHome");

// Convertir le résultat de la recherche dans le type approprié
HumanTaskManagerHome taskHome =
           (HumanTaskManagerHome)javax.rmi.PortableRemoteObject.narrow
           (result,HumanTaskManagerHome.class);

...
//Accéder à l'interface distante du bean session
HumanTaskManager task = taskHome.create();

...
//Appeler les fonctions métier exposées par l'interface de service
task.callTask(tkiid,input);
```

Accès à l'interface locale du bean session

Une application client EJB de processus métier ou de tâches manuelles accède à l'interface locale du bean session par le biais de l'interface home locale du bean.

task_context

Le bean session peut être soit le bean session BusinessFlowManager pour les applications de processus, soit le bean session HumanTaskManager pour les applications de tâche manuelle.

task_procedure

1. Ajoutez à l'interface locale du bean session une référence pointant vers le descripteur de déploiement d'applications. Ajoutez la référence à l'un des fichiers suivants :
   - Le fichier `application-client.xml` pour une application client Java Platform, Enterprise Edition (Java EE)
   - Le fichier `web.xml` pour une application Web
   - Le fichier `ejb-jar.xml` pour une application Enterprise JavaBeans (EJB)

   La référence à l'interface home locale des applications de processus est illustrée dans l'exemple suivant :

   ```
   <ejb-local-ref>
    <ejb-ref-name>ejb/LocalBusinessFlowManagerHome</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>com.ibm.bpe.api.LocalBusinessFlowManagerHome</local-home>
    <local>com.ibm.bpe.api.LocalBusinessFlowManager</local>
   </ejb-local-ref>
   ```

   La référence à l'interface home locale des applications de tâche est illustrée dans l'exemple suivant :

```
<ejb-local-ref>
 <ejb-ref-name>ejb/LocalHumanTaskManagerHome</ejb-ref-name>
 <ejb-ref-type>Session</ejb-ref-type>
 <local-home>com.ibm.task.api.LocalHumanTaskManagerHome</local-home>
 <local>com.ibm.task.api.LocalHumanTaskManager</local>
</ejb-local-ref>
```

Si vous utilisez WebSphere Integration Developer pour ajouter la référence EJB au descripteur de déploiement, la liaison de la référence EJB est automatiquement créée lors du déploiement de l'application. Pour plus d'informations concernant l'ajout de références EJB, consultez la documentation WebSphere Integration Developer.

2. Localisez l'interface home locale du bean session dans l'interface JNDI (Java Naming and Directory Interface).

   L'exemple suivant illustre cette étape pour une application de processus :

   ```
   // Obtenir le contexte JNDI initial par défaut
   InitialContext initialContext = new InitialContext();

   // Lookup the local home interface of the BusinessFlowManager bean

    LocalBusinessFlowManagerHome processHome =
         (LocalBusinessFlowManagerHome)initialContext.lookup
         ("java  :comp/env/ejb/LocalBusinessFlowManagerHome");
   ```

   L'interface home locale du bean session contient une méthode de création pour les objets EJB. Cette méthode renvoie l'interface locale du bean session.

3. Accédez à l'interface locale du bean session.

   L'exemple suivant illustre cette étape pour une application de processus :

   ```
   LocalBusinessFlowManager process = processHome.create();
   ```

   L'accès au bean session ne garantit pas que l'appelant puisse effectuer toutes les actions sur un certain processus ; l'appelant doit être également autorisé à effectuer l'action. Lorsqu'une instance du bean session est créée, elle est associée à un contexte du bean session. Le contexte contient l'ID principal de l'appelant, la liste d'appartenance au groupe, et indique si l'appelant est titulaire d'un des rôles Java EE Business Process Choreographer. Le contexte est utilisé pour vérifier l'autorisation de l'appelant pour chaque appel, même lorsque la sécurité administrative n'est pas configurée. Si la sécurité administrative n'est pas configurée, la valeur de l'ID principal de l'appelant est `UNAUTHENTICATED`.

4. Appelez les fonctions métier exposées par l'interface de service.

   L'exemple suivant illustre cette étape pour une application de processus :

   ```
   process.initiate("MyProcessModel",input);
   ```

   Les appels venant des applications sont exécutés comme des transactions. Une transaction est établie et terminée de l'une des façons suivantes :

   - Automatiquement par WebSphere Application Server (le descripteur de déploiement spécifie TX_REQUIRED).
   - De manière explicite par l'application. Vous pouvez regrouper les appels d'application à l'intérieur d'une seule transaction :

     ```
     // Obtenir l'interface de transaction utilisateur
     UserTransaction transaction=
            (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

     // Commencer une transaction
     transaction.begin();

     // Appels d'applications ...

     // En cas d'aboutissement, valider la transaction
     transaction.commit();
     ```

Tip : Afin d'éviter tout blocage de la base de données, évitez d'exécuter en parallèle des instructions similaires à la suivante :

```
// Obtain user transaction interface
UserTransaction transaction=
       (UserTransaction)initialContext.lookup("java:comp/UserTransaction");

transaction.begin();

//read the activity instance
process.getActivityInstance(aiid);
//claim the activity instance
process.claim(aiid);

transaction.commit();
```

La méthode getActivityInstance ainsi que d'autres opérations de lecture définissent un verrou en lecture. Dans cet exemple, un verrou en lecture sur l'instance d'activité est mis à niveau vers un verrou U sur l'instance d'activité. Ceci peut provoquer un blocage de la base de données lorsque ces transactions sont exécutées en parallèle

Exemple

Voici un exemple illustrant les étapes 2 à 4 pour une application de tâche.

```
//Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

//Lookup the local home interface of the HumanTaskManager bean
LocalHumanTaskManagerHome taskHome =
        (LocalHumanTaskManagerHome)initialContext.lookup
        ("java:comp/env/ejb/LocalHumanTaskManagerHome");

...
//Access the local interface of the session bean
LocalHumanTaskManager task =
taskHome.create();

...
//Call the business functions exposed by the service interface
task.callTask(tkiid,input);
```

Développement d'applications pour les processus métier

Un processus métier est un ensemble d'activités de nature professionnelle qui sont appelées dans un ordre spécifique pour atteindre un objectif professionnel. Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour des actions typiques sur des processus.

task_context

Un processus métier peut être soit un microflux, soit un processus de longue durée :

- Les microflux sont des processus métier de courte durée exécutés de manière synchrone. Après un court moment, le résultat est renvoyé à l'appelant.
- Les processus interruptibles de longue durée sont exécutés en tant que séquences d'activités chaînées. L'utilisation de certaines constructions dans un processus engendre des interruptions dans le flux de processus, notamment l'appel d'une tâche manuelle, d'un service utilisant une liaison synchrone ou encore l'utilisation d'activités automatiques.

Les branches parallèles du processus sont généralement accessibles de manière asynchrone, ce qui signifie que les activités des branches parallèles sont exécutées simultanément. En fonction du type et du paramètre de transaction de l'activité, une activité peut être exécutée au sein de sa propre transaction.

Tâches associées

«Gestion du cycle de vie d'un processus métier», à la page 451
Une instance de processus est créé lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

«Traitement des activités humaines», à la page 460
Les activités manuelles sont attribuées aux différentes personnes de votre organisation par l'intermédiaire des tâches élémentaires. Au démarrage d'un processus, des éléments de travail sont créés pour les propriétaires potentiels.

«Traitement d'un flux de travaux par une seule personne», à la page 462
Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Ce type de flux de travaux ne comporte pas de chemins d'accès parallèles. Les API initiateAndClaimFirst et completeAndClaimSuccessor prennent en charge le traitement de ce type de flux de travaux. Cet exemple illustre l'implémentation d'un flux de travaux exécutés par une seule personne à l'aide d'un flux de pages côté client.

«Envoi d'un message à une activité en attente», à la page 464
Les activités de messages entrants (également appelées activités de réception, onMessage dans des activités de sélection, onEvent dans les gestionnaires d'événements) peuvent être utilisées pour synchroniser un processus d'exécution avec des événements du "monde extérieur". Par exemple, la réception d'un courrier électronique provenant d'un client en réponse à une demande d'informations peut correspondre à ce type d'événement.

«Gestion des événements», à la page 465
L'ensemble d'un processus métier et chacune de ses portées peuvent être associés à des gestionnaires d'événements qui sont appelés si l'événement associé se produit. Les gestionnaires d'événements sont similaires aux activités de réception ou de sélection en cela qu'un processus peut fournir des opérations de service Web à l'aide de gestionnaires d'événements.

«Analyse des résultats d'un processus», à la page 466
Un processus peut afficher des opérations de services Web modélisées sous forme d'opérations WSDL (Web Services Description Language) asynchrones ou de type requête-réponse. Les résultats des processus interruptibles à interface unidirectionnelle ne peuvent être obtenus par la méthode getOutputMessage, car ces processus ne produisent pas de résultat. Cependant, vous pouvez interroger le contenu des variables.

«Réparation d'activités», à la page 467
Un processus de longue durée peut contenir des activités dont l'exécution est également longue. Ces activités peuvent rencontrer des erreurs non interceptées et se trouver ainsi à l'état arrêté. Une activité à l'état actif peut également sembler ne plus répondre. Dans les deux cas, un administrateur de processus peut intervenir sur l'activité de plusieurs manières afin que la navigation du processus puisse se poursuivre.

Référence associée

«Rôles nécessaires pour effectuer des actions sur des instances de processus», à la page 449
L'accès à l'interface BusinessFlowManager ne garantit pas que l'appelant puisse effectuer toutes les actions sur un processus donné. L'appelant doit être également autorisé à effectuer l'action en étant titulaire d'un rôle approprié.

«Rôles nécessaires pour effectuer des actions sur les activités de processus métier», à la page 450

L'accès à l'interface BusinessFlowManager ne garantit pas que l'appelant puisse effectuer toutes les actions sur une activité donnée. L'appelant doit être également autorisé à effectuer l'action en étant titulaire d'un rôle approprié.

«Interface BusinessFlowManagerService», à la page 473
L'interface BusinessFlowManagerService permet l'accès aux fonctions de processus métier pouvant être appelées par une application client.

Rôles nécessaires pour effectuer des actions sur des instances de processus

L'accès à l'interface BusinessFlowManager ne garantit pas que l'appelant puisse effectuer toutes les actions sur un processus donné. L'appelant doit être également autorisé à effectuer l'action en étant titulaire d'un rôle approprié.

Le tableau suivant indique les actions qu'un rôle spécifique peut effectuer sur une instance de processus.

Action	**Rôle principal de l'appelant**		
	Lecteur	**Initiateur**	**Administrateur**
createMessage	x	x	x
createWorkItem			x
delete			x
deleteWorkItem			x
forceTerminate			x
getActiveEventHandlers	x		x
getActivityInstance	x		x
getAllActivities	x		x
getAllWorkItems	x		x
getClientUISettings	x	x	x
getCustomProperties	x	x	x
getCustomProperty	x	x	x
getCustomPropertyNames	x	x	x
getFaultMessage	x	x	x
getInputClientUISettings	x	x	x
getInputMessage	x	x	x
getOutputClientUISettings	x	x	x
getOutputMessage	x	x	x
getProcessInstance	x	x	x
getVariable	x	x	x
getWaitingActivities	x	x	x
getWorkItems	x		x
restart			x
resume			x
setCustomProperty		x	x
setVariable			x
suspend			x
transferWorkItem			x

Remarque : Si l'administration des processus est restreinte aux administrateurs système, l'administration basée sur les instances est désactivée. Cela signifie que les actions d'administration sur les processus, les portées et les activités sont limitées aux utilisateurs du rôle BPESystemAdministrator. En outre, la lecture, l'affichage et la surveillance d'une instance de processus (intégralement ou partiellement) ne peuvent être effectués que par les utilisateurs des rôles BPESystemAdministrator ou BPESystemMonitor. Pour plus d'informations sur ce rôle d'administration, voir doc/bpc/cnot_instance_based_admin.dita.

Rôles nécessaires pour effectuer des actions sur les activités de processus métier

L'accès à l'interface BusinessFlowManager ne garantit pas que l'appelant puisse effectuer toutes les actions sur une activité donnée. L'appelant doit être également autorisé à effectuer l'action en étant titulaire d'un rôle approprié.

Le tableau suivant indique les actions qu'un rôle spécifique peut effectuer sur une instance d'activité.

Action	**Rôle principal de l'appelant**				
	Lecteur	**Editeur**	**Propriétaire potentiel**	**Propriétaire**	**Administrateur**
cancelClaim				x	x
claim			x		x
complete				x	x
createMessage	x	x	x	x	x
createWorkItem					x
deleteWorkItem					x
forceComplete					x
forceRetry					x
getActivityInstance	x	x	x	x	x
getAllWorkItems	x	x	x	x	x
getClientUISettings	x	x	x	x	x
getCustomProperties	x	x	x	x	x
getCustomProperty	x	x	x	x	x
getCustomPropertyNames	x	x	x	x	x
getFaultMessage	x	x	x	x	x
getFaultNames	x	x	x	x	x
getInputMessage	x	x	x	x	x
getOutputMessage	x	x	x	x	x
getVariable	x	x	x	x	x
getVariableNames	x	x	x	x	x
getInputVariableNames	x	x	x	x	x
getOutputVariableNames	x	x	x	x	x
getWorkItems	x	x	x	x	x
setCustomProperty		x		x	x

Action	Rôle principal de l'appelant				
	Lecteur	**Editeur**	**Propriétaire potentiel**	**Propriétaire**	**Administrateur**
setFaultMessage		x		x	x
setOutputMessage		x		x	x
setVariable					x
transferWorkItem				x Réservé au propriétaires ou administrateurs potentiels	x

Remarque : Si l'administration des processus est restreinte aux administrateurs système, l'administration basée sur les instances est désactivée. Cela signifie que les actions d'administration sur les processus, les portées et les activités sont limitées aux utilisateurs du rôle BPESystemAdministrator. En outre, la lecture, l'affichage et la surveillance d'une instance de processus (intégralement ou partiellement) ne peuvent être effectués que par les utilisateurs des rôles BPESystemAdministrator ou BPESystemMonitor. Pour plus d'informations sur ce rôle d'administration, voir doc/bpc/cnot_instance_based_admin.dita.

Gestion du cycle de vie d'un processus métier

Une instance de processus est créé lorsqu'une méthode API de Business Process Choreographer pouvant démarrer un processus est appelée. La navigation de l'instance de processus continue jusqu'à ce que l'ensemble de ses activités se trouvent à l'état final. Plusieurs actions peuvent être entreprises sur l'instance de processus afin de gérer son cycle de vie.

task_context

Des exemples fournis illustrent la façon dont vous pourriez développer des applications pour les actions de cycle de vie typiques sur les processus.

Tâches associées

«Démarrage de processus métier»
La façon dont un processus métier est démarré varie selon que le processus est un microflux ou un processus de longue durée. Le service qui démarre le processus est également important par rapport à la façon dont un processus est démarré ; le processus peut avoir soit un service de démarrage unique, soit plusieurs services de démarrage.

«Mise en suspens et reprise d'un processus métier», à la page 457
Vous pouvez mettre en suspens une instance de processus de niveau supérieur de longue durée pendant qu'elle est en cours d'exécution, puis la relancer ultérieurement.

«Redémarrage d'un processus métier», à la page 458
Vous pouvez redémarrer une instance de processus se trouvant à l'état terminé, arrêté, échoué ou compensé.

«Arrêt d'une instance de processus», à la page 459
Parfois une instance de processus de niveau supérieur à l'état irrécupérable doit être arrêtée.

«Suppression d'instances de processus», à la page 459
Les instances de processus terminées sont automatiquement supprimées de la base de données de Business Process Choreographer si la propriété correspondante est définie pour le modèle de processus dans le modèle de processus. Vous pouvez choisir de conserver les instances de processus dans votre base de données, par exemple, pour rechercher des données relatives aux instances de processus qui ne sont pas consignées dans le journal d'audit. Cependant, les données d'instance de processus stockées n'ont pas seulement une incidence sur l'espace disque et les performances mais elles empêchent la création d'instances de processus utilisant les mêmes valeurs d'ensembles de corrélations. Vous devez par conséquent supprimer régulièrement les données d'instances de processus de la base de données.

Démarrage de processus métier :

La façon dont un processus métier est démarré varie selon que le processus est un microflux ou un processus de longue durée. Le service qui démarre le processus est également important par rapport à la façon dont un processus est démarré ; le processus peut avoir soit un service de démarrage unique, soit plusieurs services de démarrage.

task_context

Des exemples sont fournis pour illustrer la façon dont vous pouvez développer des applications pour les scénarios de démarrage habituels des microflux et des processus longue durée.

Tâches associées

«Exécution d'un microflux contenant un service de démarrage unique»
Un microflux peut être lancé par une activité de réception ou une activité de sélection. Le service de démarrage est unique si le microflux démarre avec une activité de réception ou lorsque l'activité de sélection n'a qu'une définition onMessage.

«Exécution d'un microflux contenant un service de démarrage non unique», à la page 454
Un microflux peut être lancé par une activité de réception ou une activité de sélection. Le service de démarrage n'est pas unique si le microflux démarre avec une activité de sélection possédant plusieurs définitions onMessage.

«Démarrage d'un processus de longue durée contenant un service de démarrage unique», à la page 455
Si le service de démarrage est unique, vous pouvez utiliser la méthode de déclenchement et transmettre le nom du modèle de processus en tant que paramètre. C'est le cas lorsque le processus de longue durée démarre avec une activité de sélection ou de réception unique et lorsque l'activité de sélection unique n'a qu'une définition onMessage.

«Démarrage d'un processus de longue durée contenant un service de démarrage non unique», à la page 456
Un processus de longue durée peut être lancé par le biais de plusieurs activités de sélection ou de réception déclenchantes. Vous pouvez utiliser la méthode de déclenchement pour lancer le processus. Si le service de démarrage n'est pas unique, par exemple si le processus démarre avec plusieurs activités de réception ou de sélection ou avec une activité de sélection possédant plusieurs définitions onMessage, vous devez identifier le service à appeler.

Exécution d'un microflux contenant un service de démarrage unique :

Un microflux peut être lancé par une activité de réception ou une activité de sélection. Le service de démarrage est unique si le microflux démarre avec une activité de réception ou lorsque l'activité de sélection n'a qu'une définition onMessage.

task_context

Si le microflux implémente une opération de requête-réponse, c'est à dire si le processus contient une réponse, vous pouvez utiliser la méthode d'appel pour exécuter le processus transmettant le nom de modèle de processus comme paramètre d'appel.

Si le microflux est une opération unidirectionnelle, exécutez le processus via la méthode sendMessage. Cette méthode n'est pas traitée dans l'exemple.

task_procedure

1. OptionalColonSymbol Répertoriez les modèles de processus pour trouver le nom du processus que vous voulez exécuter.

   Cette étape est facultative si vous connaissez déjà le nom du processus.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
("PROCESS_TEMPLATE.EXECUTION_MODE =
       PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_MICROFLOW",
 "PROCESS_TEMPLATE.NAME",
  new Integer(50),
  (TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles classés pouvant être lancés par la méthode d'appel.

2. Lancez le processus avec un message de sortie du type approprié.

   Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

```
ProcessTemplateData template = processTemplates[0];
//create a message for the single starting receive activity
ClientObjectWrapper input = process.createMessage
                           (template.getID(),
                            template.getInputMessageTypeName());
DataObject myMessage = null;
if ( input.getObject()!= null && input.getObject()
instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the strings in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}

//run the process
ClientObjectWrapper output = process.call(template.getName(), input);
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
  myOutput  = (DataObject)output.getObject();
  int order = myOutput.getInt("OrderNo");
}
```

   Cette opération crée une instance du modèle de processus, CustomerTemplate, et transfère quelques données client. L'opération renvoie uniquement lorsque le processus est terminé. Le résultat du processus, OrderNo, est renvoyé à l'appelant.

Exécution d'un microflux contenant un service de démarrage non unique :

Un microflux peut être lancé par une activité de réception ou une activité de sélection. Le service de démarrage n'est pas unique si le microflux démarre avec une activité de sélection possédant plusieurs définitions onMessage.

task_context

Si le microflux implémente une opération de requête-réponse, c'est à dire si le processus contient une réponse, vous pouvez utiliser la méthode d'appel pour exécuter le processus transmettant l'ID du service de démarrage comme paramètre d'appel.

Si le microflux est une opération unidirectionnelle, exécutez le processus via la méthode sendMessage. Cette méthode n'est pas traitée dans l'exemple.

task_procedure

1. OptionalColonSymbol Répertoriez les modèles de processus pour trouver le nom du processus que vous voulez exécuter.

   Cette étape est facultative si vous connaissez déjà le nom du processus.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
("PROCESS_TEMPLATE.EXECUTION_MODE =
       PROCESS_TEMPLATE.EXECUTION_MODE.EXCECUTION_MODE_MICROFLOW",
 "PROCESS_TEMPLATE.NAME",
  new Integer(50),
  (TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles classés pouvant être lancés en tant que microflux.

2. Déterminez le service de démarrage à appeler.

   Cet exemple utilise le premier modèle trouvé.

   ```
   ProcessTemplateData template = processTemplates[0];
   ActivityServiceTemplateData[] startActivities =
          process.getStartActivities(template.getID());
   ```

3. Lancez le processus avec un message de sortie du type approprié.

   Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

   ```
   ActivityServiceTemplateData activity = startActivities[0];
   //create a message for the service to be called
   ClientObjectWrapper input =
         process.createMessage(activity.getServiceTemplateID(),
                               activity.getActivityTemplateID(),
                               activity.getInputMessageTypeName());
   DataObject myMessage = null;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //set the strings in the message, for example, a customer name
     myMessage.setString("CustomerName", "Smith");
   }
   //run the process
   ClientObjectWrapper output = process.call(activity.getServiceTemplateID(),
                                             activity.getActivityTemplateID(),
                                             input);
   //check the output of the process, for example, an order number
   DataObject myOutput = null;
   if ( output.getObject() != null && output.getObject() instanceof DataObject )
   {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
   }
   ```

   Cette opération crée une instance du modèle de processus, CustomerTemplate, et transfère quelques données client. L'opération renvoie uniquement lorsque le processus est terminé. Le résultat du processus, OrderNo, est renvoyé à l'appelant.

Démarrage d'un processus de longue durée contenant un service de démarrage unique :

Si le service de démarrage est unique, vous pouvez utiliser la méthode de déclenchement et transmettre le nom du modèle de processus en tant que paramètre. C'est le cas lorsque le processus de longue durée démarre avec une activité de sélection ou de réception unique et lorsque l'activité de sélection unique n'a qu'une définition onMessage.

task_procedure

1. OptionalColonSymbol Répertoriez les modèles de processus pour trouver le nom du processus que vous voulez lancer.

   Cette étape est facultative si vous connaissez déjà le nom du processus.

   ```
   ProcessTemplateData[] processTemplates = process.queryProcessTemplates
     ("PROCESS_TEMPLATE.EXECUTION_MODE =
          PROCESS_TEMPLATE.EXECUTION_MODE.EXECUTION_MODE_LONG_RUNNING",
      "PROCESS_TEMPLATE.NAME",
       new Integer(50),
       (TimeZone)null);
   ```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles classés pouvant être lancés par la méthode de déclenchement.

2. Lancez le processus avec un message de sortie du type approprié.

   Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message. Si vous spécifiez un nom d'instance de processus, il ne doit pas commencer par un trait de soulignement. Si aucun nom d'instance de processus n'est spécifié, l'identifiant d'instance de processus (PIID) au format chaîne est utilisé en tant que nom.

```
ProcessTemplateData template = processTemplates[0];
//create a message for the single starting receive activity
ClientObjectWrapper input = process.createMessage
                            (template.getID(),
                            template.getInputMessageTypeName());
DataObject myMessage = null;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //set the strings in the message, for example, a customer name
  myMessage.setString("CustomerName", "Smith");
}
//start the process
PIID piid = process.initiate(template.getName(), "CustomerOrder", input);
```

   Cette opération crée une instance, CustomerOrder, et transfère quelques données client. Lorsque le processus démarre, l'opération renvoie à l'appelant l'identifiant objet de la nouvelle instance de processus.

   L'initiateur de l'instance de processus est défini pour l'appelant de la requête. Cette personne reçoit un élément de travail pour l'instance de processus. Les administrateurs du processus, les lecteurs et les éditeurs de l'instance de processus sont déterminés et reçoivent des éléments de travail pour l'instance de processus. Les instances d'activité suivie sont déterminées. Elles sont lancées automatiquement, ou, si ce sont des activités manuelles, de réception ou de sélection, des éléments de travail sont créés pour les éventuels propriétaires.

Démarrage d'un processus de longue durée contenant un service de démarrage non unique :

Un processus de longue durée peut être lancé par le biais de plusieurs activités de sélection ou de réception déclenchantes. Vous pouvez utiliser la méthode de déclenchement pour lancer le processus. Si le service de démarrage n'est pas unique, par exemple si le processus démarre avec plusieurs activités de réception ou de sélection ou avec une activité de sélection possédant plusieurs définitions onMessage, vous devez identifier le service à appeler.

task_procedure

1. OptionalColonSymbol Répertoriez les modèles de processus pour trouver le nom du processus que vous voulez lancer.

   Cette étape est facultative si vous connaissez déjà le nom du processus.

```
ProcessTemplateData[] processTemplates = process.queryProcessTemplates
  ("PROCESS_TEMPLATE.EXECUTION_MODE =
       PROCESS_TEMPLATE.EXECUTION_MODE.EXECUTION_MODE_LONG_RUNNING",
   "PROCESS_TEMPLATE.NAME",
    new Integer(50),
    (TimeZone)null);
```

Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles classés pouvant être lancés en tant que processus de longue durée.

2. Déterminez le service de démarrage à appeler.

```
ProcessTemplateData template = processTemplates[0];
ActivityServiceTemplateData[] startActivities =
        process.getStartActivities(template.getID());
```

3. Lancez le processus avec un message de sortie du type approprié.

   Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message. Si vous spécifiez un nom d'instance de processus, il ne doit pas commencer par un trait de soulignement. Si aucun nom d'instance de processus n'est spécifié, l'identifiant d'instance de processus (PIID) au format chaîne est utilisé en tant que nom.

```
ActivityServiceTemplateData activity = startActivities[0];
//créez un message pour le service à appeler
ClientObjectWrapper input = process.createMessage
                           (activity.getServiceTemplateID(),
                            activity.getActivityTemplateID(),
                            activity.getInputMessageTypeName());
DataObject myMessage = null;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //définissez les chaînes dans le message, par exemple, un nom de client
  myMessage.setString("CustomerName", "Smith");
}
//démarrez le processus
PIID piid = process.sendMessage(activity.getServiceTemplateID(),
                            activity.getActivityTemplateID(),
                            input);
```

   Cette opération crée une instance et transfère quelques données client. Lorsque le processus démarre, l'opération renvoie à l'appelant l'identifiant objet de la nouvelle instance de processus.

   L'initiateur de l'instance de processus est défini pour l'appelant de la requête et reçoit un élément de travail pour l'instance de processus. Les administrateurs du processus, les lecteurs et les éditeurs de l'instance de processus sont déterminés et reçoivent des éléments de travail pour l'instance de processus. Les instances d'activité suivie sont déterminées. Elles sont lancées automatiquement, ou, si ce sont des activités manuelles, de réception ou de sélection, des éléments de travail sont créés pour les éventuels propriétaires.

Mise en suspens et reprise d'un processus métier :

Vous pouvez mettre en suspens une instance de processus de niveau supérieur de longue durée pendant qu'elle est en cours d'exécution, puis la relancer ultérieurement.

task_prereq

task_context

Vous pouvez avoir besoin de mettre en suspens une instance de processus, par exemple, pour pouvoir configurer l'accès à un système dorsal qui est utilisé ultérieurement dans le processus. Une fois que les conditions prérequises pour le processus sont remplies, vous pouvez reprendre l'instance de processus. Vous

pouvez également souhaiter interrompre un processus afin de résoudre un problème engendrant l'échec de l'instance de processus, puis le reprendre une fois le problème résolu.

Pour qu'une instance de processus puisse être mise en suspens, elle doit se trouver à l'état exécution en cours ou échec en cours. L'appelant doit être un administrateur de processus ou un administrateur système. Toutefois, si Business Flow Manager utilise le mode d'autorisation pour l'administration alternative du processus, qui restreint l'administration des processus aux administrateurs système, seuls les appelants du rôle BPESystemAdministrator peuvent effectuer cette action.

task_procedure

1. Obtenez le processus en cours d'exécution, CustomerOrder, que vous souhaitez mettre en suspens.

   ```
   ProcessInstanceData processInstance =
                       process.getProcessInstance("CustomerOrder");
   ```

2. Mettez l'instance de processus en suspens.

   ```
   PIID piid = processInstance.getID();
   process.suspend( piid );
   ```

   Cette action suspend l'instance de processus de niveau supérieur spécifiée. L'instance de processus passe à l'état mis en suspens. Dans cet état, les activités lancées peuvent être terminées, mais aucune nouvelle activité n'est activée. Les sous-processus dont l'attribut autonomy est défini sur `enfant` (child) sont également suspendus, s'ils étaient en cours d'exécution, en état d'échec, terminés ou en cours de compensation. Les tâches en ligne et les tâches autonomes qui sont associées à cette instance de processus ne sont pas suspendues.

3. Reprenez l'instance de processus.

   ```
   process.resume( piid );
   ```

   Cette action met l'instance de processus et ses sous processus dans l'état où ils se trouvaient avant d'être mis en suspens.

Redémarrage d'un processus métier :

Vous pouvez redémarrer une instance de processus se trouvant à l'état terminé, arrêté, échoué ou compensé.

task_context

Le redémarrage d'une instance de processus est similaire au démarrage initial d'une instance de processus. Toutefois, lorsqu'une instance de processus est redémarrée, l'identifiant de l'instance de processus est connu et le message d'entrée pour l'instance est disponible.

Si le processus possède plusieurs activités de réception ou activités de sélection (également appelées activités de choix de réception) capables de créer l'instance de processus, tous les messages qui appartiennent à ces activités sont utilisés pour le redémarrage de l'instance de processus. Si l'une de ces activités implémentent une opération de requête-réponse, la réponse est envoyée à nouveau lors du survol de l'activité de réponse associée.

L'appelant doit être un administrateur de processus ou un administrateur système. Toutefois, si Business Flow Manager utilise le mode d'autorisation pour l'administration alternative du processus, qui restreint l'administration des processus aux administrateurs système, seuls les appelants du rôle

BPESystemAdministrator peuvent effectuer cette action.

task_procedure

1. Obtenez le processus que vous souhaitez redémarrer.

```
ProcessInstanceData processInstance =
                    process.getProcessInstance("CustomerOrder");
```

2. Redémarrez l'instance de processus.

```
PIID piid = processInstance.getID();
process.restart( piid );
```

   Cette action redémarrez l'instance de processus spécifiée.

Arrêt d'une instance de processus :

Parfois une instance de processus de niveau supérieur à l'état irrécupérable doit être arrêtée.

task_context

Pour effectuer cette action, l'appelant doit être un administrateur de processus ou un administrateur système. Toutefois, si Business Flow Manager utilise le mode d'autorisation pour l'administration alternative du processus, qui restreint l'administration des processus aux administrateurs système, seuls les appelants du rôle BPESystemAdministrator peuvent effectuer cette action.

Etant donné qu'une instance de processus se termine immédiatement, sans attendre l'arrêt de sous processus ou d'activités en cours, vous ne devez effectuer cette action que dans des cas exceptionnels.

task_procedure

1. Procédez à l'extraction de l'instance de processus devant être arrêtée.

```
ProcessInstanceData processInstance =
       process.getProcessInstance("CustomerOrder");
```

2. Arrêtez l'instance de processus.

   Si vous arrêtez une instance de processus, vous pouvez arrêter l'instance de processus avec ou sans compensation.

   Pour arrêter l'instance de processus avec compensation :

```
PIID piid = processInstance.getID();
process.forceTerminate(piid, CompensationBehaviour.INVOKE_COMPENSATION);
```

   Pour arrêter l'instance de processus sans compensation :

```
PIID piid = processInstance.getID();
process.forceTerminate(piid);
```

   Si vous arrêtez l'instance de processus avec compensation, la compensation du processus est exécutée comme si une erreur était survenue sur la portée de niveau supérieur. Si vous arrêtez l'instance de processus sans compensation, l'instance de processus est arrêtée aussitôt sans attendre que les activités en cours, les tâches à effectuer ou les tâches d'appel intégrées ne se terminent normalement.

   Les applications démarrées par le processus et les tâches autonomes liées au processus ne sont pas arrêtées par la requête d'arrêt forcé. Si l'arrêt de ces applications est prévu, vous devez ajouter à l'application du processus les déclarations destinées à mettre fin explicitement aux applications initiées par le processus.

Suppression d'instances de processus :

Les instances de processus terminées sont automatiquement supprimées de la base de données de Business Process Choreographer si la propriété correspondante est définie pour le modèle de processus dans le modèle de processus. Vous pouvez choisir de conserver les instances de processus dans votre base de données, par exemple, pour rechercher des données relatives aux instances de processus qui ne sont pas consignées dans le journal d'audit. Cependant, les données d'instance de processus stockées n'ont pas seulement une incidence sur l'espace disque et les performances mais elles empêchent la création d'instances de processus utilisant les mêmes valeurs d'ensembles de corrélations. Vous devez par conséquent supprimer régulièrement les données d'instances de processus de la base de données.

task_context

Pour supprimer une instance de processus, vous devez traiter les droits d'administrateur et l'instance de processus doit être une instance de processus de niveau supérieur.

L'exemple suivant montre comment supprimer toutes les instances de processus terminées.

task_procedure

1. Répertoriez les instances de processus qui sont terminées.

```
QueryResultSet result =
     process.query("DISTINCT PROCESS_INSTANCE.PIID",
                   "PROCESS_INSTANCE.STATE =
                            PROCESS_INSTANCE.STATE.STATE_FINISHED",
                    (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie un ensemble de résultats de requête qui répertorie les instances de processus terminées.

2. Supprimez les instances de processus terminées.

```
while (result.next() )
{
   PIID piid = (PIID) result.getOID(1);
   process.delete(piid);
}
```

   Cette action supprime l'instance de processus sélectionnée et ses tâches en ligne de la base de données.

Traitement des activités humaines

Les activités manuelles sont attribuées aux différentes personnes de votre organisation par l'intermédiaire des tâches élémentaires. Au démarrage d'un processus, des éléments de travail sont créés pour les propriétaires potentiels.

task_context

Lorsqu'une activité manuelle est activée, une instance d'activité et une tâche à effectuer associée sont créées en même temps. Le traitement de l'activité manuelle et la gestion de l'élément de travail sont délégués à l'application Human Task Manager. Toute modification d'état au niveau de l'instance d'activité est reflétée dans l'instance d'activité et inversement.

Un propriétaire potentiel réclame l'activité. Cette personne est responsable de fournir les informations pertinentes et de mener l'activité à terme.

task_procedure

1. Répertoriez les activités appartenant à une personne connectée et qui sont prêtes à être traitées :

```
QueryResultSet result =
     process.query("ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_READY AND
                    ACTIVITY.KIND = ACTIVITY.KIND.KIND_STAFF AND
                    WORK_ITEM.REASON =
                         WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                    (String)null, (Integer)null, (TimeZone)null);
```

   Cette action renvoie un ensemble de résultats de requête contenant les activités pouvant être gérées par la personne connectée.

2. Réclamez l'activité à gérer :

```
if (result.size() > 0)
{
 result.first();
 AIID aiid = (AIID) result.getOID(1);
 ClientObjectWrapper input = process.claim(aiid);
 DataObject activityInput = null ;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
  {
    activityInput = (DataObject)input.getObject();
    // lire les valeurs
    ...
  }
}
```

   Une fois l'activité réclamée, le message d'entrée de l'activité est renvoyé.

3. Une fois la gestion de l'activité terminée, terminez celle-ci. L'activité peut se terminer correctement, ou produire un message d'erreur. En cas de succès de l'activité, un message de sortie est transmis. En cas d'échec de l'activité, celle-ci est mise en état d'échec ou d'arrêt et un message d'erreur est transmis. Vous devez créer les messages appropriés pour ces opérations. Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

   a. Pour terminer l'activité correctement, créez un message de sortie.

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //définir les parties du message d'erreur, par exemple un numéro d'ordre
  myMessage.setInt("OrderNo", 4711);
}

//fin de l'activité
process.complete(aiid, output);
```

      Cette opération définit un message de sortie contenant le numéro de commande.

   b. Pour terminer l'activité lorsque se produit une erreur, créez un message d'erreur.

```
//retrieve the faults modeled for the human task activity
List faultNames = process.getFaultNames(aiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
```

```
      process.createMessage(aiid, faultNames.get(0));

// set the parts in your fault message, for example, an error number
DataObject myMessage = null ;
if ( myFault.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //set the parts in the message, for example, a customer name
  myMessage.setInt("error",1304);
}

process.complete(aiid, myFault,(String)faultNames.get(0) );
```

Cette action définit l'activité comme ayant l'état en échec ou arrêté. Si le paramètre **continueOnError** de l'activité contenue dans le modèle de processus est défini sur la valeur true, l'activité est mise en état d'échec et la navigation se poursuit. Si le paramètre **continueOnError** est défini sur false et que l'erreur n'est pas traitée dans la portée environnante, l'activité est mise à l'état arrêté. Lorsque l'activité se trouve dans cet état, elle peut être réparée via un arrêt ou un redémarrage forcé.

Concepts associés

Comportement de poursuite après erreur des activités et processus métier
Lorsque vous définissez un processus métier, vous pouvez indiquer le comportement à adopter lorsqu'une erreur est détectée et qu'aucun gestionnaire d'erreur n'est défini pour cette erreur. Vous pouvez utiliser le paramètre **Poursuivre après erreur** lors de la définition du processus pour indiquer que ce processus doit être arrêté lorsque l'erreur survient.

Traitement d'un flux de travaux par une seule personne

Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Ce type de flux de travaux ne comporte pas de chemins d'accès parallèles. Les API initiateAndClaimFirst et completeAndClaimSuccessor prennent en charge le traitement de ce type de flux de travaux. Cet exemple illustre l'implémentation d'un flux de travaux exécutés par une seule personne à l'aide d'un flux de pages côté client.

task_context

Un flux de travaux exécuté par une seule personne est également appelé *flux de pages* ou *flux d'écrans*. Il existe deux types de flux de pages :

- Les flux de pages côté client, où la navigation entre les différentes pages est effectuée à l'aide de la technologie côté client (par exemple, un formulaire Lotus Forms sur plusieurs pages).
- Les flux de pages côté serveur sont réalisés à l'aide d'un processus métier et d'un ensemble de tâches manuelles modélisées de sorte que les tâches suivantes soient affectées à la même personne.

Les flux de pages côté serveur sont plus puissants que les flux de pages côté client, mais ils utilisent davantage de ressources serveur en vue de leur traitement. Par conséquent, il est recommandé de n'utiliser ce type de flux de travaux que dans les cas suivants :

- Vous devez appeler des services entre les étapes effectuées dans une interface utilisateur (par exemple, pour extraire ou mettre à jour des données).
- Certaines de vos exigences d'audit requièrent l'écriture d'événements CEI après interaction avec une interface utilisateur.

C'est notamment le cas lorsqu'un flux de travaux exécuté par une seule personne correspond au processus de commande dans une librairie en ligne, où l'acheteur effectue une série d'actions pour commander un livre. Cette série d'actions peut être implémentée comme une série d'activités manuelles (tâches à accomplir). Si l'acheteur décide de commander plusieurs livres, cela équivaut à démarrer un processus de commande et à réclamer l'activité de tâche manuelle suivante.

L'API initiateAndClaimFirst démarre le flux de pages, c'est-à-dire qu'elle démarre le processus spécifié et réclame la première activité de tâche manuelle dans la séquence d'activités. Elle renvoie des informations sur l'activité réclamée, notamment sur le message d'entrée sur lequel travailler.

L'API completeAndClaimSuccessor termine alors l'activité de tâche manuelle et réclame l'activité suivante dans la même instance de processus pour la personne connectée. L'API renvoie ensuite les informations sur l'activité réclamée suivante, y compris le message d'entrée à traiter. L'activité suivante étant disponible dans la même transaction que celle de l'activité terminée, le comportement transactionnel de toutes les activités de tâche manuelle du modèle de processus doit être définir sur `participates`.

Comparez cet exemple avec celui qui utilise à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

task_procedure

1. Démarrez le processus de commande des livres et réclamez la première activité dans la séquence d'activités. Lancez le processus avec un message de sortie du type approprié. Lorsque vous créez le message, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message. Si vous spécifiez un nom d'instance de processus, il ne doit pas commencer par un trait de soulignement. Si aucun nom d'instance de processus n'est spécifié, l'identifiant d'instance de processus (PIID) au format chaîne est utilisé en tant que nom.
   a. Récupérez le modèle de processus pour créer un message d'entrée du type approprié.

      ```
      ProcessTemplateData template = process.getProcessTemplate("CustomerOrder");
      ClientObjectWrapper input = process.createMessage(template.getID(),
                                  template.getInputMessageTypeName());
      DataObject myMessage = null;
      if ( input.getObject()!= null && input.getObject() instanceof DataObject )
      {
        myMessage = (DataObject)input.getObject();
        //set the strings in the message, for example, a customer name
        myMessage.setString("CustomerName", "Smith");
      }
      ```

   b. Démarrez le processus et réclamez la première activité de tâche manuelle.

      ```
      InitiateAndClaimFirstResult result =
        process.initiateAndClaimFirst("CustomerOrder", "MyOrderProcess", input);
      AIID aiid = result.getAIID();
      ClientObjectWrapper input = result.getInputMessage();
      ```

   Lorsque la première activité est réclamée, le message d'entrée et l'ID de l'activité réclamée sont renvoyés.
2. Une fois la gestion de l'activité terminée, terminez celle-ci et réclamez l'activité suivante.

   Pour terminer l'activité, un message de sortie est créé. Lorsque vous créez le message de sortie, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //définir les parties du message d'erreur, par exemple un numéro d'ordre
  myMessage.setInt("OrderNo", 4711);
}

//Fin de l'activité et réclamation de la suivante
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

Cette opération définit un message de sortie contenant le numéro de commande et réclame l'activité suivante de la séquence. Si `AutoClaim` est défini pour les activités de succession et que plusieurs chemins d'accès peuvent être utilisés, toutes les activités de succession sont réclamées et une activité aléatoire est renvoyée en tant qu'activité suivante. Si aucune activité de succession supplémentaire ne peut être affectée à cet utilisateur, la valeur `Null` est renvoyée.

Si le processus contient des chemins parallèles pouvant être suivis, que ces chemins contiennent des activités manuelles et que l'utilisateur connecté est le propriétaire potentiel de plusieurs de ces activités, une activité aléatoire est automatiquement réclamée et renvoyée comme activité suivante.

3. Traitement de l'activité suivante.

```
String name = successor.getActivityName();

ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // lire les valeurs
  ...
}

aiid = successor.getAIID();
```

4. Poursuivez à l'étape 2 pour terminer l'activité.

Tâches associées

«Traitement par une seule personne d'un flux de travaux contenant des tâches manuelles», à la page 500
Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Cet exemple montre comment implémenter un flux de travaux exécutés par une seule personne à l'aide d'un flux de pages côté serveur. Les API de Business Flow Manager et Human Task Manager sont toutes les deux utilisées pour traiter le flux de travaux.

Envoi d'un message à une activité en attente

Les activités de messages entrants (également appelées activités de réception, onMessage dans des activités de sélection, onEvent dans les gestionnaires d'événements) peuvent être utilisées pour synchroniser un processus d'exécution avec des événements du "monde extérieur". Par exemple, la réception d'un courrier électronique provenant d'un client en réponse à une demande d'informations peut correspondre à ce type d'événement.

task_context

Vous pouvez utiliser des tâches d'origine pour envoyer le message à l'activité.

task_procedure

1. Répertoriez les modèles de services d'activité attendant un message de l'utilisateur connecté dans une instance de processus avec un ID d'instance de processus spécifique.

   ```
   ActivityServiceTemplateData[] services = process.getWaitingActivities(piid);
   ```

2. Envoyez un message au premier service en attente.

   On suppose que le premier service est celui que vous souhaitez servir. L'appelant doit être un démarreur potentiel de l'activité recevant le message ou un administrateur de l'instance de processus.

   ```
   VTID vtid = services[0].getServiceTemplateID();
   ATID atid = services[0].getActivityTemplateID();
   String inputType = services[0].getInputMessageTypeName();

   // créer un message pour le service à appeler
     ClientObjectWrapper message =
           process.createMessage(vtid,atid,inputMessageTypeName);
     DataObject myMessage = null;
     if ( message.getObject()!= null && message.getObject() instanceof DataObject )
     {
       myMessage = (DataObject)message.getObject();
       //set the strings in the message, for example, chocolate is to be ordered
       myMessage.setString("Order", "chocolate");
     }

     // envoi du message à l'activité en attente
     process.sendMessage(vtid, atid, message);
   }
   ```

   Cette opération envoie le message spécifié au service d'activité en attente et transfère certaines données de commande.

   Vous pouvez également spécifier l'identifiant de l'instance de processus afin de veiller à ce que le message soit envoyé à l'instance de processus spécifiée. Si l'identifiant de l'instance de processus n'est pas spécifié, le message est envoyé au service d'activité et à l'instance de processus identifiée par les valeurs de corrélation du message. Si l'identifiant de l'instance de processus est spécifié, l'instance de processus trouvée à l'aide des valeurs de corrélation est vérifiée afin de veiller à ce qu'elle possède bien l'identifiant de l'instance de processus spécifiée.

Gestion des événements

L'ensemble d'un processus métier et chacune de ses portées peuvent être associés à des gestionnaires d'événements qui sont appelés si l'événement associé se produit. Les gestionnaires d'événements sont similaires aux activités de réception ou de sélection en cela qu'un processus peut fournir des opérations de service Web à l'aide de gestionnaires d'événements.

task_context

Vous pouvez appeler un gestionnaire d'événements autant de fois que vous le souhaitez tant que la portée correspondante est en cours d'exécution. Par ailleurs, plusieurs instances d'un gestionnaire d'événements peuvent être activées en même temps.

Le fragment de code suivant montre comment obtenir les gestionnaires d'événements actifs pour une instance de processus donnée et comment envoyer un message d'entrée.

task_procedure

1. Déterminez les données de l'identifiant d'instance de processus et répertoriez les gestionnaires d'événements actifs pour le processus.

```
ProcessInstanceData processInstance =
       process.getProcessInstance( "CustomerOrder2711");
EventHandlerTemplateData[] events = process.getActiveEventHandlers(
                                       processInstance.getID() );
```

2. Envoyez le message d'entrée.

   Cet exemple utilise le premier gestionnaire d'événements trouvé.

```
EventHandlerTemplateData event = null;
if ( events.length > 0 )
{
    event = events[0];

    // créer un message pour le service à appeler
    ClientObjectWrapper input = process.createMessage(
    event.getID(), event.getInputMessageTypeName());

    if (input.getObject() != null && input.getObject() instanceof DataObject )
    {
       DataObject inputMessage = (DataObject)input.getObject();
       // définir le contenu du message, par exemple, un nom de client, numéro de commande
       inputMessage.setString("CustomerName", "Smith");
       inputMessage.setString("OrderNo", "2711");

        // envoyer le message
       process.sendMessage( event.getProcessTemplateName(),
                            event.getPortTypeNamespace(),
                            event.getPortTypeName(),
                            event.getOperationName(),
     input );
     }
 }
```

   Cette opération envoie le message spécifié au gestionnaire d'événements actif pour le processus.

Analyse des résultats d'un processus

Un processus peut afficher des opérations de services Web modélisées sous forme d'opérations WSDL (Web Services Description Language) asynchrones ou de type requête-réponse. Les résultats des processus interruptibles à interface unidirectionnelle ne peuvent être obtenus par la méthode getOutputMessage, car ces processus ne produisent pas de résultat. Cependant, vous pouvez interroger le contenu des variables.

task_context

Les résultats du processus ne sont stockés dans la base de données que si le modèle de processus dont dérive l'instance de processus ne spécifie pas une suppression automatique des instances de processus dérivées.

task_procedure

Analysez les résultats des processus. Vérifiez par exemple le numéro de commande.

```
QueryResultSet result = process.query
                       ("PROCESS_INSTANCE.PIID",
                        "PROCESS_INSTANCE.NAME = 'CustomerOrder' AND
                         PROCESS_INSTANCE.STATE =
                                 PROCESS_INSTANCE.STATE.STATE_FINISHED",
                         (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  PIID piid = (PIID) result.getOID(1);
  ClientObjectWrapper output = process.getOutputMessage(piid);
  DataObject myOutput = null;
  if ( output.getObject() != null && output.getObject() instanceof DataObject )
  {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
  }
}
```

Réparation d'activités

Un processus de longue durée peut contenir des activités dont l'exécution est également longue. Ces activités peuvent rencontrer des erreurs non interceptées et se trouver ainsi à l'état arrêté. Une activité à l'état actif peut également sembler ne plus répondre. Dans les deux cas, un administrateur de processus peut intervenir sur l'activité de plusieurs manières afin que la navigation du processus puisse se poursuivre.

task_context

L'API de Business Process Choreographer propose les méthodes de réparation d'activité forceRetry et forceComplete. Plusieurs exemples illustrent l'ajout et la réparation d'actions pour des activités de vos applications.

Tâches associées

«Forcer une activité à se terminer»
Les activités situées dans des processus de longue durée rencontrent parfois des erreurs. Si ces erreurs ne sont pas interceptes par un gestionnaire d'erreurs dans la portée et si le modèle d'activité associé spécifie que l'activité doit s'arrêter lorsqu'une erreur se produit, l'activité est mise à l'état arrêté de manière à pouvoir être réparée. Dans cet état, vous pouvez forcer l'activité à se terminer.

«Nouvelle tentative d'exécution d'une activité arrêtée», à la page 469
Si une activité d'un processus de longue durée rencontre une erreur non interceptée dans la portée et si le modèle d'activité associé spécifie que l'activité doit s'arrêter lorsqu'une erreur se produit, l'activité est mise à l'état arrêté de manière à pouvoir être réparée. Vous pouvez tenter d'exécuter à nouveau l'activité.

«Réparation des activités arrêtées en raison de l'échec d'une évaluation de jointure, de fin de boucle ou de décompte», à la page 470
Les activités peuvent être arrêtées si une exception survient lors de l'évaluation d'une condition de jointure ou de fin de boucle, ou de l'évaluation d'une valeur de décompte forEach. L'administrateur décide de ne par relancer l'activité, par exemple, parce que l'évaluation risque d'échouer à nouveau. Dans ce cas, les valeurs correctes pour l'expression peuvent être fournies à l'aide de l'API EJB de Business Process Choreographer EJB API ce qui permet à la navigation du processus de se poursuivre.

«Mise à jour des ensembles de corrélations associés aux activités arrêtées», à la page 472
Les ensembles de corrélations permettent de prendre en charge la collaboration avec état entre des services Web. Dans ce cas, les valeurs correctes pour l'expression peuvent être fournies à l'aide de l'API EJB de Business Process Choreographer pour permettre la poursuite de la navigation du processus.

Forcer une activité à se terminer :

Les activités situées dans des processus de longue durée rencontrent parfois des erreurs. Si ces erreurs ne sont pas interceptes par un gestionnaire d'erreurs dans la portée et si le modèle d'activité associé spécifie que l'activité doit s'arrêter lorsqu'une erreur se produit, l'activité est mise à l'état arrêté de manière à pouvoir être réparée. Dans cet état, vous pouvez forcer l'activité à se terminer.

task_context

Vous pouvez également forcer l'achèvement des activités en cours d'exécution si, par exemple, une activité ne répond pas.

Des exigences supplémentaires existent pour certains types d'activités.

Activités manuelles
Vous pouvez transmettre des paramètres dans l'appel forcer à terminer, comme le message qui aurait du être envoyé ou l'erreur qui aurait dû être détectée.

Activités de script
Vous ne pouvez pas transmettre de paramètres dans l'appel forcer à terminer. Cependant, vous devez définir les variables qui doivent être réparées.

Activités d'appel
Vous pouvez également forcer l'achèvement des activités d'appel appelant un service asynchrone qui n'est pas un sous-processus si ces activités sont

dans l'état en cours d'exécution. Vous pouvez en avoir besoin, par exemple, si le service asynchrone est appelé et ne répond pas.

task_procedure

1. Répertoriez les activités arrêtées qui se trouvent à l'état arrêté.

```
QueryResultSet result =
     process.query("DISTINCT ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                    PROCESS_INSTANCE.NAME='CustomerOrder'",
                    (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie les activités arrêtées pour l'instance de processus CustomerOrder.

2. Achevez l'activité ; une activité manuelle arrêtée, par exemple.

   Dans cet exemple, un message de sortie est transmis.

```
if (result.size() > 0)
{
   result.first();
   AIID aiid = (AIID) result.getOID(1);
   ActivityInstanceData activity = process.getActivityInstance(aiid);
  ClientObjectWrapper output =
        process.createMessage(aiid, activity.getOutputMessageTypeName());
  DataObject myMessage = null;
  if ( output.getObject()!= null && output.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)output.getObject();
     //set the parts in your message, for example, an order number
     myMessage.setInt("OrderNo", 4711);
   }

   boolean continueOnError = true;
   process.forceComplete(aiid, output, continueOnError);
}
```

   Cette action effectue l'activité. Si une erreur survient, le paramètre **continueOnError** détermine l'action à entreprendre en cas d'erreur lors du traitement de la requête forceComplete.

   Dans l'exemple, **continueOnError** est vrai. Cette valeur signifie que si une erreur se produit, l'activité est mise à l'état d'échec. L'erreur se propage aux portées de l'activité jusqu'à ce qu'elle soit gérée ou que la portée du processus soit atteinte. Le processus est alors mis à l'état d'échec en cours avant d'atteindre finalement l'état d'échec.

Concepts associés

Comportement de poursuite après erreur des activités et processus métier
Lorsque vous définissez un processus métier, vous pouvez indiquer le comportement à adopter lorsqu'une erreur est détectée et qu'aucun gestionnaire d'erreur n'est défini pour cette erreur. Vous pouvez utiliser le paramètre **Poursuivre après erreur** lors de la définition du processus pour indiquer que ce processus doit être arrêté lorsque l'erreur survient.

Nouvelle tentative d'exécution d'une activité arrêtée :

Si une activité d'un processus de longue durée rencontre une erreur non interceptée dans la portée et si le modèle d'activité associé spécifie que l'activité doit s'arrêter lorsqu'une erreur se produit, l'activité est mise à l'état arrêté de manière à pouvoir être réparée. Vous pouvez tenter d'exécuter à nouveau l'activité.

task_context

Vous pouvez définir des variables utilisées par l'activité. à l'exception des activités de script, vous pouvez également transmettre des paramètres dans l'appel forcer la nouvelle tentative, comme le message qui était attendu par l'activité.

task_procedure

1. Répertoriez les activités arrêtées.

```
QueryResultSet result =
      process.query("DISTINCT ACTIVITY.AIID",
                    "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                     PROCESS_INSTANCE.NAME='CustomerOrder'",
                     (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie les activités arrêtées pour l'instance de processus CustomerOrder.

2. Tentez à nouveau d'exécuter l'activité, une activité manuelle, par exemple.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);
  ActivityInstanceData activity = process.getActivityInstance(aiid);
  ClientObjectWrapper input =
        process.createMessage(aiid, activity.getOutputMessageTypeName());
  DataObject myMessage = null;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
    {
      myMessage = (DataObject)input.getObject();
      //set the strings in your message, for example, chocolate is to be ordered
      myMessage.setString("OrderNo", "chocolate");
    }

   boolean continueOnError = true;
   process.forceRetry(aiid, input, continueOnError);
}
```

   Cette opération tente à nouveau d'exécuter l'activité. Si une erreur se produit, le paramètre **continueOnError** détermine l'action à entreprendre en cas d'erreur lors du traitement de la requête forceRetry.

   Dans l'exemple, **continueOnError** est vrai. Cela signifie que si une erreur se produit durant le traitement de la requête forceRetry, l'activité est mise en état échec. L'erreur se propage aux portées de l'activité jusqu'à ce qu'elle soit gérée ou que la portée du processus soit atteinte. Le processus est alors mis à l'état d'échec en cours, puis un gestionnaire d'erreur au niveau du processus est exécuté avant que le processus n'atteigne l'état d'échec.

Concepts associés

Comportement de poursuite après erreur des activités et processus métier
Lorsque vous définissez un processus métier, vous pouvez indiquer le comportement à adopter lorsqu'une erreur est détectée et qu'aucun gestionnaire d'erreur n'est défini pour cette erreur. Vous pouvez utiliser le paramètre **Poursuivre après erreur** lors de la définition du processus pour indiquer que ce processus doit être arrêté lorsque l'erreur survient.

Réparation des activités arrêtées en raison de l'échec d'une évaluation de jointure, de fin de boucle ou de décompte :

Les activités peuvent être arrêtées si une exception survient lors de l'évaluation d'une condition de jointure ou de fin de boucle, ou de l'évaluation d'une valeur de décompte forEach. L'administrateur décide de ne par relancer l'activité, par

exemple, parce que l'évaluation risque d'échouer à nouveau. Dans ce cas, les valeurs correctes pour l'expression peuvent être fournies à l'aide de l'API EJB de Business Process Choreographer EJB API ce qui permet à la navigation du processus de se poursuivre.

task_context

Vous pouvez définir la valeur d'une condition de jointure pour tout type d'activité, la valeur d'une condition de fin de boucle d'une activité while ou repeat-until. Vous pouvez également définir les valeurs de décompte initial et final et le nombre maximal de branches terminées pour chaque activité forEach. La valeur que vous définissez pour les branches terminées dépend de la définition de l'activité forEach dans le modèle de processus. Si une condition d'exit anticipée est indiquée dans le modèle, définissez une valeur pour le nombre maximal de branches terminées. Si une condition d'exit anticipée n'est pas indiquée, paramétrez la valeur du nombre maximal de branches terminées sur `null`.

L'exemple suivant montre comment définir la valeur d'une condition de fin de boucle.

task_procedure

1. Affichez la liste des activités qui se sont arrêtées en raison de l'échec de l'évaluation d'une condition de fin de boucle.

```
QueryResultSet result = process.query(
  "DISTINCT ACTIVITY.AIID",
  "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
   ACTIVITY.STOP_REASON = ACTIVITY.STOP_REASON.STOP_REASON_IMPLEMENTATION_FAILED AND
   (ACTIVITY.KIND = ACTIVITY.KIND.KIND_WHILE OR
      ACTIVITY.KIND = ACTIVITY.KIND.KIND_REPEAT_UNTIL) AND
   PROCESS_INSTANCE.NAME='CustomerOrder'",
   (String)null, (Integer)null, (TimeZone)null );
```

   De même, vous pouvez afficher la liste des activités qui se sont arrêtées en raison de l'échec de l'évaluation d'une condition de jointure ou d'un décompte forEach.

   - Pour une condition de jointure qui a échoué, utilisez l'expression suivante :

     ```
     ACTIVITY.STOP_REASON.STOP_REASON_ACTIVATION_FAILED
     ```

   - Pour un décompte forEach qui a échoué, utilisez l'expression suivante :

```
ACTIVITY.STOP_REASON.STOP_REASON_IMPLEMENTATION_FAILED AND
(ACTIVITY.KIND = ACTIVITY.KIND.KIND_FOR_EACH_SERIAL OR
   ACTIVITY.KIND = ACTIVITY.KIND.KIND_FOR_EACH_PARALLEL)
```

   Cette action renvoie les activités pour l'instance de processus CustomerOrder qui a échoué en raison de l'échec de l'évaluation d'une condition de fin de boucle.

2. Fournissez la valeur de la condition de fin de boucle, par exemple, true.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);

  process.forceLoopCondition(aiid, true);
}
```

   Cette action paramètre la valeur de la condition de fin de boucle pour l'activité sur true et la navigation de l'instance de processus se poursuit.

De manière similaire, vous pouvez paramétrer la valeur d'une condition de jointure (`process.forceJoinCondition(aiid, true);`) ou les valeurs de décomptes d'activité forEach (`process.forceForEachCounterValues(aiid, 1, 5, new Integer(2));`).

Mise à jour des ensembles de corrélations associés aux activités arrêtées :

Les ensembles de corrélations permettent de prendre en charge la collaboration avec état entre des services Web. Dans ce cas, les valeurs correctes pour l'expression peuvent être fournies à l'aide de l'API EJB de Business Process Choreographer pour permettre la poursuite de la navigation du processus.

task_context

Une activité qui est à l'état arrêté peut nécessiter la mise à jour de l'ensemble de corrélations qui lui est associé pour une des raisons suivantes :

- Une exception s'est produite lors de l'évaluation de l'ensemble de corrélations. L'ensemble de corrélations doit être initialisé alors qu'il l'est déjà.
- Une exception s'est produite lors de l'évaluation de l'ensemble de corrélations. L'ensemble de corrélations doit être initialisé mais ses valeurs ne sont pas définies. C'est le cas, par exemple, lorsqu'une activité d'initialisation a été ignorée.
- Il est nécessaire de réessayer l'activité. Si l'ensemble de corrélations est initialisé par l'activité, son initialisation peut être annulée ou il peut être modifié avant l'appel de la méthode forceRetry.
- Il est nécessaire de terminer l'activité. Si l'ensemble de corrélations est initialisé par l'activité, son initialisation peut être annulée ou il peut être modifié avant l'appel de la méthode forceComplete.

Vous pouvez extraire les instances d'ensemble de corrélations d'une instance de processus ou d'activité. L'exemple suivant montre comment initialiser ou annuler l'initialisation d'instances d'ensemble de corrélations.

task_procedure

1. Répertoriez les activités arrêtées qui se trouvent à l'état arrêté.

```
QueryResultSet result =
     process.query("DISTINCT ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                    PROCESS_INSTANCE.NAME='CustomerOrder'",
                    (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie les activités arrêtées pour l'instance de processus CustomerOrder.

2. Extrayez les instances d'ensemble de corrélations définies pour l'activité.

```
AIID aiid = null;

List correlationSet = null;

if (result.size() > 0)
{
   result.first();
   AIID aiid = (AIID) result.getOID(1);

   ActivityInstanceData activity = process.getActivityInstance(aiid);

   correlationSet = process.getCorrelationSetInstances
                    (aiid, activity.getInputMessageTypeName());
}
```

3. Annulez l'initialisation de l'ensemble de corrélations, par exemple, MonEnsembleCorrélations.

```
for ( int i=0; i<correlationSet.size(); i++ )
{
  CorrelationSetInstanceData correlationSetInstance =
            (CorrelationSetInstanceData)correlationSet.get(i);

  if ( correlationSetInstance.isInitialized() &&
       correlationSetInstance.getCorrelationSetName().equals("MonEnsembleCorrélations") )
  {
    process.uninitializeCorrelationSet
       ( activity.getProcessInstanceID(), correlSetInstance.getCorrelationSetName() );
  }
}
```

   Cette action annule l'initialisation de l'ensemble de corrélations MonEnsembleCorrélations.

4. Initialisez l'ensemble de corrélations, par exemple, MonEnsembleCorrélations. Dans cet exemple, une propriété avec une valeur de type chaîne de l'ensemble de corrélations est définie.

```
for ( int i=0; i<correlationSet.size(); i++ )
{
  CorrelationSetInstanceData correlationSetInstance =
            (CorrelationSetInstanceData)correlationSet.get(i);

  if ( correlationSetInstance.getCorrelationSetName().equals("MonEnsembleCorrélations") )
  {
    List correlationSetProperties =
                  correlationSetInstance.getCorrelationSetProperties();
    for ( int j=0; j<correlationSetProperties.size(); j++ )
    {
      CorrelationPropertyInstanceData property =
                (CorrelationPropertyInstanceData)correlationSetProperties.get(j);

      if ( property.getPropertyName().equals("MaPropriété") )
      {
        property.setValue("NouvelleValeur");

        process.initializeCorrelationSet
           ( activity.getProcessInstanceID(), correlationSetInstance );
      }
    }
  }
}
```

   Cette action initialise la propriété ayant une valeur de type chaîne MaPropriété dans l'ensemble de corrélations MonEnsembleCorrélations.

Interface BusinessFlowManagerService

L'interface BusinessFlowManagerService permet l'accès aux fonctions de processus métier pouvant être appelées par une application client.

Les méthodes pouvant être appelées par l'intermédiaire de l'interface BusinessFlowManagerService varient selon l'état du processus ou de l'activité et des droits d'accès de l'utilisateur de l'application qui contient la méthode. Les méthodes principales de manipulation des objets de processus métier sont répertoriées dans cette rubrique. Plus plus d'information sur ces méthodes et d'autres méthodes fournies par l'interface BusinessFlowManagerService, consultez Javadoc dans le package com.ibm.bpe.api.

Modèles de processus

Le modèle de processus est un exemple de processus mis à niveau, déployé et installé contenant la spécification d'un processus métier. Vous pouvez l'instancier et

le démarrer en lançant les demandes appropriées, par exemple sendMessage(). L'exécution de l'instance de processus est automatiquement gérée par le serveur.

Tableau 60. Méthodes API pour les modèles de processus

Méthode	Description
getProcessTemplate	Extrait le modèle de processus spécifié.
queryProcessTemplates	Extrait des modèles de processus stockés dans la base de données.

Traitement d'instances

Les méthodes API suivantes sont liées au démarrage des instances de processus.

Tableau 61. Méthodes API liées au démarrage des instances de processus

Méthode	Description
call	Crée et exécute un microflux.
callWithReplyContext	Crée et exécute un microflux avec un service à démarrage unique ou un processus longue durée provenant du modèle de processus spécifié. L'appel attend le renvoi du résultat en mode asynchrone.
callWithUISettings	Crée et exécute un processus et renvoie le message de sortie et les paramètres de l'interface utilisateur (UI) du client.
initiate	Crée et exécute une instance de processus et démarre son traitement. Cette méthode est adaptée aux processus longue durée. Vous pouvez également appliquer cette méthode aux microflux destinés à être déclenchés, puis laissés sans surveillance.
initiateAndSuspend	Crée une instance de processus, mais suspend son traitement.
initiateAndClaimFirst	Crée une instance de processus et réclame la première tâche manuelle en ligne.
sendMessage	Envoie le message spécifié au service d'activité et à l'instance de processus spécifiés. Si une instance de processus possédant les mêmes valeurs que l'ensemble de corrélations n'existe pas, celle-ci est créée. Le processus peut posséder des services de démarrage uniques ou non.
getStartActivities	Renvoie des informations sur les activités qui peuvent démarrer une instance de processus à partir du modèle de processus spécifié.
getActivityServiceTemplate	Extrait le modèle de service de l'activité spécifiée.

Tableau 62. Méthodes API pour le contrôle du cycle de vie des instances de processus

Méthode	Description
suspend	Met en suspens l'exécution d'une instance de processus de longue durée, de niveau supérieur se trouvant à l'état d'échec en cours ou d'exécution en cours.
resume	Reprend l'exécution d'une instance de processus de longue durée, de niveau supérieur se trouvant à l'état mis en suspens.
restart	Redémarre une instance de processus de longue durée, de niveau supérieur se trouvant à l'état terminé, échoué ou arrêté.
forceTerminate	Termine l'instance de processus de niveau supérieur spécifiée, ses sous-processus avec autonomie enfant et ses activités en cours d'exécution, réclamées, ou en attente
delete	Supprime l'instance de processus de niveau supérieur spécifiée et ses sous-processus avec autonomie enfant.
query	Extrait à partir de la base de données les propriétés correspondant aux critères de recherche.
queryEntities	Utilise les tables de requête pour extraite de la base de données les propriétés correspondant aux critères de recherche.
getWaitingActivities	Renvoie des informations sur les activités qui attendent un message afin que leur traitement puisse continuer.
migrate	Migre une instance de processus vers la version plus récente spécifiée du modèle de processus associé.

Activités

Pour les activités d'appel, vous pouvez spécifier dans le modèle de processus que ces activités doivent continuer dans des situations d'erreur. Si l'indicateur continueOnError est défini sur false et qu'une erreur non gérée survient, l'activité passe à l'état arrêté. L'administrateur du processus peut ensuite réparer l'activité . L'indicateur continueOnError et les fonctions de réparation associées peuvent être utilisés, par exemple, pour un processus de longue durée où les activités d'appel échouent occasionnellement mais où l'effort requis pour modéliser la compensation et la gestion des erreurs est trop important.

Les méthodes suivantes sont disponibles pour l'utilisation et la réparation des activités.

Tableau 63. Méthodes API pour le contrôle du cycle de vie des instances d'activité

Méthode	Description
claim	Réclame une instance d'activité prête pour permettre à un utilisateur d'utiliser l'activité.
cancelClaim	Annule la réclamation de l'instance d'activité.

Tableau 63. Méthodes API pour le contrôle du cycle de vie des instances d'activité (suite)

Méthode	Description
complete	Termine l'instance d'activité.
completeAndClaimSuccessor	Effectue une instance d'activité et demande la suivante dans la même instance de processus pour l'utilisateur connecté.
forceComplete	Force l'exécution des éléments suivants : • Une instance d'activité se trouvant à l'état en cours d'exécution ou arrêté. • Une activité de tâche manuelle se trouvant à l'état prêt ou réclamé. • Une attente d'attente se trouvant à l'état en attente.
forceRetry	Force la répétition des éléments suivants : • Une instance d'activité se trouvant à l'état en cours d'exécution ou arrêté. • Une activité de tâche manuelle se trouvant à l'état prêt ou réclamé.
forceNavigate, forceForEach, forceLoop, forceJoin	Ces méthodes forcent la navigation d'une activité arrêtée.
skip	Ignore le traitement de l'activité.
jump	Permet de passer d'une activité à une autre.
query	Extrait à partir de la base de données les propriétés correspondant aux critères de recherche.
queryEntities	Utilise les tables de requête pour extraite de la base de données les propriétés correspondant aux critères de recherche.

Variables et propriétés personnalisées

L'interface fournit une méthode get et une méthode set pour l'extraction et la définition de valeurs pour les variables. Vous pouvez aussi associer les propriétés mentionnées aux instances de processus et d'activité et les en extraire. Le noms de propriétés personnalisées et des valeurs doivent être de type java.lang.String.

Tableau 64. Méthodes API pour les variables et les propriétés personnalisées

Méthode	Description
getVariable	Extrait la variable spécifiée.
setVariable	Définit la variable spécifiée.
getCustomProperty	Extrait la propriété personnalisée indiquée de l'activité ou instance de processus indiqué.
getCustomProperties	Extrait les propriétés personnalisées de l'activité ou de l'instance de processus indiquée.
getCustomPropertyNames	Extrait les noms des propriétés personnalisées pour l'instance d'activité ou de processus spécifiée.

Tableau 64. Méthodes API pour les variables et les propriétés personnalisées (suite)

Méthode	Description
setCustomProperty	Stocke les valeurs spécifiques aux propriétés personnalisées correspondant à l'instance d'activité ou de processus spécifiée.

Développement d'applications pour des tâches manuelles

Une tâche représente le moyen par lequel des composants appellent des humains en tant que services ou par lequel des humains appellent des services. Des exemples d'applications typiques pour des tâches manuelles sont fournis.

task_context

Pour plus d'informations concernant l'API de Human Task Manager, voir la documentation Java dans le package com.ibm.task.api.

Tâches associées

«Démarrage d'une tâche d'appel qui appelle une interface synchrone», à la page 479
Une tâche d'appel est associée au composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Ne démarrez une tâche d'appel synchrone que si le composant SCA associé peut être appelé de manière synchrone.

«Démarrage d'une tâche d'appel qui appelle une interface asynchrone», à la page 480
Une tâche d'appel est associée au composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Ne démarrez une tâche d'appel asynchrone que si le composant SCA associé peut être appelé de manière asynchrone.

«Création et lancement d'une instance de tâche», à la page 481
Ce scénario indique comment créer une instance de modèle de tâche permettant de définir une tâche de collaboration (également appelée *tâche manuelle* et de démarrer l'instance de tâche.

«Traitement des tâches à effectuer ou des tâches de collaboration», à la page 481
Les tâches à effectuer (également appelées *tâches de participation* dans l'API) ou les tâches de collaboration (également appelées *tâches manuelles* dans l'API) sont attribuées à diverses personnes de votre organisation par le biais des éléments de travail. Les tâches à effectuer et leurs éléments de travail associés sont créés, par exemple, lorsqu'un processus navigue jusqu'à une activité manuelle.

«Mise en suspens et reprise d'une instance de tâche», à la page 483
Vous pouvez interrompre les instances de tâche de collaboration (également appelées *tâches manuelles* dans l'API) ou les instances de tâche à effectuer (également appelées *tâches de participation* dans l'API).

«Analyse des résultats d'une tâche», à la page 484
Une tâche à effectuer (également appelée tâche de *participation* dans l'API) ou une tâche de collaboration (également appelée *tâche manuelle* dans l'API) fonctionne de manière asynchrone. Si un gestionnaire de réponses est indiqué lors du démarrage d'une tâche, le message de sortie est automatiquement retourné à la fin de celle-ci. Dans le cas contraire, le message doit être extrait explicitement.

«Arrêt d'une instance de tâche», à la page 484
Il s'avère parfois nécessaire pour quelqu'un disposant de droits d'administration d'arrêter une instance de tâche dans un état irrécupérable. Etant donné qu'une instance de tâche s'arrête instantanément, cette opération ne doit être exécutée que dans des situations exceptionnelles.

«Suppression d'instances de tâche», à la page 485
Les instances de tâche ne sont automatiquement supprimées que lorsqu'elles sont terminées, à condition que cela soit spécifié dans le modèle de tâche associé dont dérivent les instances. Cet exemple montre comment supprimer toutes les instances de tâche qui sont terminées mais ne sont pas supprimées automatiquement.

«Libération d'une tâche réclamée», à la page 485
Lorsqu'un propriétaire potentiel réclame une tâche, il lui incombe de mener la tâche à son terme. Toutefois, certaines tâches réclamées doivent être libérées pour afin qu'un autre propriétaire potentiel puisse la réclamer à son tour.

«Gestion des tâches élémentaires», à la page 486
Durant la durée de vie d'une instance d'activité ou de tâche, l'ensemble des personnes associées à l'objet peut changer, par exemple, si une personne est en congé, si de nouvelles personnes sont engagées ou si la charge de travail doit être redistribuée. Pour autoriser ces modifications, vous devez développer des applications afin de créer, supprimer ou transférer les tâches élémentaires.

«Création de modèles de tâche et d'instances de tâche à l'exécution», à la page 487
Un outil de modélisation, comme WebSphere Integration Developer, permet habituellement de compiler des modèles de tâche. Vous installez les modèles de tâche dans WebSphere Process Server et créez des instances à partir de ces modèles en utilisant, par exemple, Business Process Choreographer Explorer. Cependant, vous pouvez également créer des instances de tâche manuelle ou de participation lors de l'exécution.

Référence associée

«Interface HumanTaskManagerService», à la page 494
L'interface HumanTaskManagerService permet l'accès aux fonctions relatives aux tâches pouvant être appelées par des clients locaux ou distants.

Démarrage d'une tâche d'appel qui appelle une interface synchrone

Une tâche d'appel est associée au composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Ne démarrez une tâche d'appel synchrone que si le composant SCA associé peut être appelé de manière synchrone.

task_context

Un tel composant SCA peut, par exemple, être implémenté en tant que microflux ou en tant que classe Java simple.

Ce scénario crée une instance d'un modèle de tâche et transmet certaines données client. La tâche reste à l'état actif jusqu'à la fin de l'opération bidirectionnelle. Le résultat de la tâche, OrderNo, est renvoyé à l'appelant.

task_procedure

1. OptionalColonSymbol Répertoriez les modèles de tâche pour trouver le nom de la tâche d'appel que vous voulez exécuter.

   Cette étape est facultative si vous connaissez déjà le nom de la tâche.

   ```
   TaskTemplate[] taskTemplates = task.queryTaskTemplates
   ("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
    "TASK_TEMPL.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles d'origine classés.

2. Créez un message d'entrée pour le type approprié.

   ```
   TaskTemplate template = taskTemplates[0];

   // créer un a message pour la tâche sélectionnée
   ClientObjectWrapper input = task.createInputMessage( template.getID());
   DataObject myMessage = null ;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //définir les parties du message, par exemple, un nom de client
     myMessage.setString("CustomerName", "Smith");
   }
   ```

3. Créez la tâche et exécutez la tâche de façon synchrone.

   Pour qu'une tâche s'exécute de façon synchrone, il doit s'agir d'une opération bidirectionnelle. L'exemple utilise la méthode createAndCallTask pour créer et exécuter la tâche.

```
ClientObjectWrapper output = task.createAndCallTask( template.getName(),
                                                     template.getNamespace(),
                                                     input);
```

4. Analysez le résultat de la tâche.

```
DataObject myOutput = null;
if ( output.getObject() != null && output.getObject() instanceof DataObject )
{
  myOutput  = (DataObject)output.getObject();
  int order = myOutput.getInt("OrderNo");
}
```

Démarrage d'une tâche d'appel qui appelle une interface asynchrone

Une tâche d'appel est associée au composant SCA (Service Component Architecture). Une fois la tâche démarrée, elle appelle le composant SCA. Ne démarrez une tâche d'appel asynchrone que si le composant SCA associé peut être appelé de manière asynchrone.

task_context

Un tel composant SCA peut, par exemple, être implémenté en tant que processus à long terme ou en tant qu'opération unidirectionnelle.

Ce scénario crée une instance d'un modèle de tâche et transmet certaines données client.

task_procedure

1. OptionalColonSymbol Répertoriez les modèles de tâche pour trouver le nom de la tâche d'appel que vous voulez exécuter.

   Cette étape est facultative si vous connaissez déjà le nom de la tâche.

```
TaskTemplate[] taskTemplates = task.queryTaskTemplates
  ("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_ORIGINATING",
   "TASK_TEMPL.NAME",
    new Integer(50),
    (TimeZone)null);
```

   Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles d'origine classés.

2. Créez un message d'entrée pour le type approprié.

```
TaskTemplate template = taskTemplates[0];

// créer un a message pour la tâche sélectionnée
ClientObjectWrapper input = task.createInputMessage( template.getID());
DataObject myMessage = null ;
if ( input.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)input.getObject();
  //définir les parties du message, par exemple, un nom de client
  myMessage.setString("CustomerName", "Smith");
}
```

3. Créez la tâche et exécutez-la de façon asynchrone.

   L'exemple utilise la méthode createAndStartTask pour créer et exécuter la tâche.

```
task.createAndStartTask( template.getName(),
                         template.getNamespace(),
                         input,
                         (ReplyHandlerWrapper)null);
```

Création et lancement d'une instance de tâche

Ce scénario indique comment créer une instance de modèle de tâche permettant de définir une tâche de collaboration (également appelée *tâche manuelle* et de démarrer l'instance de tâche.

task_procedure

1. OptionalColonSymbol Répertoriez les modèles de tâche pour trouver l'ID du modèle de tâche (TKTID) de la tâche de collaboration que vous voulez exécuter.

   Cette étape est facultative si vous connaissez déjà l'ID du modèle de tâche.

   ```
   TaskTemplate[] taskTemplates = task.queryTaskTemplates
   ("TASK_TEMPL.KIND=TASK_TEMPL.KIND.KIND_HUMAN",
    "TASK_TEMPL.NAME",
     new Integer(50),
     (TimeZone)null);
   ```

   Les résultats sont classés par nom. La requête renvoie un tableau contenant les 50 premiers modèles de tâche classés.

2. Créez un message d'entrée pour le type approprié.

   ```
   TaskTemplate template = taskTemplates[0];

   // créer un a message pour la tâche sélectionnée
   ClientObjectWrapper input = task.createInputMessage( template.getID());
   DataObject myMessage = null ;
   if ( input.getObject()!= null && input.getObject() instanceof DataObject )
   {
     myMessage = (DataObject)input.getObject();
     //définir les parties du message, par exemple, un nom de client
     myMessage.setString("CustomerName", "Smith");
   }
   ```

3. Création et démarrage de la tâche de collaboration (aucun gestionnaire de réponse n'est spécifié dans cet exemple).

   L'exemple utilise la méthode createAndStartTask pour créer et démarrer la tâche.

   ```
   TKIID tkiid = task.createAndStartTask( template.getName(),
                                          template.getNamespace(),
                                          input,
                                          (ReplyHandlerWrapper)null);
   ```

   Des éléments de travail sont créés pour les personnes concernées par l'instance de tâche. Un propriétaire potentiel, par exemple, peut réclamer la nouvelle instance de tâche.

4. Réclamation de l'instance de tâche.

   ```
   ClientObjectWrapper input2 = task.claim(tkiid);
   DataObject taskInput = null ;
   if ( input2.getObject()!= null && input2.getObject() instanceof DataObject )
   {
     taskInput = (DataObject)input2.getObject();
     // lire les valeurs
     ...
   }
   ```

   Une fois l'instance de tâche réclamée, le message d'entrée de la tâche est renvoyé.

Traitement des tâches à effectuer ou des tâches de collaboration

Les tâches à effectuer (également appelées *tâches de participation* dans l'API) ou les tâches de collaboration (également appelées *tâches manuelles* dans l'API) sont attribuées à diverses personnes de votre organisation par le biais des éléments de travail. Les tâches à effectuer et leurs éléments de travail associés sont créés, par exemple, lorsqu'un processus navigue jusqu'à une activité manuelle.

task_context

L'un des propriétaires potentiels réclame la tâche associée à l'élément de travail. Cette personne est responsable de fournir les informations pertinentes et de mener la tâche à terme.

task_procedure

1. Répertoriez les tâches appartenant à une personne connectée qui sont prêtes à être effectuées.

```
QueryResultSet result =
     task.query("TASK.TKIID",
                "TASK.STATE = TASK.STATE.STATE_READY AND
                (TASK.KIND = TASK.KIND.KIND_PARTICIPATING OR
                 TASK.KIND = TASK.KIND.KIND_HUMAN)AND
                 WORK_ITEM.REASON =
                   WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                 (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie un ensemble de résultats de requête contenant les tâches pouvant être effectuées par la personne connectée.

2. Réclamez la tâche à effectuer.

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  ClientObjectWrapper input = task.claim(tkiid);
  DataObject taskInput = null ;
  if ( input.getObject()!= null && input.getObject() instanceof DataObject )
  {
    taskInput = (DataObject)input.getObject();
    // lire les valeurs
    ...
  }
}
```

   Une fois la tâche réclamée, le message d'entrée de la tâche est renvoyé.

3. Une fois le travail de la tâche effectué, terminez la tâche.

   La tâche peut se terminer correctement ou par un message d'erreur. Si la tâche s'exécute correctement, un message de sortie est transmis. Si la tâche ne s'exécute pas correctement, un message d'erreur est transmis. Vous devez créer les messages appropriés pour ces opérations.

   a. Pour terminer la tâche correctement, créez un message de sortie.

```
ClientObjectWrapper output =
      task.createOutputMessage(tkiid);
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//fin de la tâche
task.complete(tkiid, output);
```

      Cette opération définit un message de sortie contenant le numéro de commande. La tâche est mise à l'état terminé.

   b. Pour terminer la tâche lorsque se produit une erreur, créez un message d'erreur.

```
//retrieve the faults modeled for the task List faultNames =
task.getFaultNames(tkiid);
ListfaultNames input = task.getFaultNames(tkiid);

//create a message of the appropriate type
ClientObjectWrapper myFault =
      task.createFaultMessage(tkiid, (String)faultNames.get(0));

// définir les parties du message d'erreur, par exemple un numéro d'erreur
DataObject myMessage = null ;
if ( myFault.getObject()!= null && input.getObject() instanceof DataObject )
{
  myMessage = (DataObject)myFault.getObject();
  //définir les parties du message, par exemple, un nom de client
  myMessage.setInt("error",1304);
}

task.complete(tkiid, (String)faultNames.get(0), myFault);
```

Cette action définit un message d'erreur qui contient le code d'erreur. La tâche est mise à l'état d'échec.

Concepts associés

Diagrammes de transition d'état pour les tâches de collaboration
Les tâches de collaboration facilitent le travail des utilisateurs lorsqu'ils sont amenés à travailler avec d'autres personnes. Durant le cycle de vie d'une tâche de collaboration, certaines interactions ne sont possibles que lorsque la tâche en question est dans un état précis, et ces interactions ont elles-mêmes une incidence sur l'état futur de la tâche.

Mise en suspens et reprise d'une instance de tâche

Vous pouvez interrompre les instances de tâche de collaboration (également appelées *tâches manuelles* dans l'API) ou les instances de tâche à effectuer (également appelées *tâches de participation* dans l'API).

task_prereq

L'instance de tâche peut se trouver à l'état prêt ou réclamé. Elle peut être transférée à un niveau supérieur. L'appelant doit être le propriétaire, l'émetteur ou l'administrateur de l'instance de tâche.

task_context

Vous pouvez mettre une instance de tâche en suspens durant son exécution. Il peut également être souhaitable d'effectuer cette opération dans le but de recueillir des informations nécessaires pour achever la tâche. Une fois ces informations disponibles, vous pouvez reprendre l'exécution de l'instance de tâche.

task_procedure

1. Obtention de la liste des tâches réclamées par l'utilisateur connecté.

   ```
   QueryResultSet result = task.query("DISTINCT TASK.TKIID",
                                      "TASK.STATE = TASK.STATE.STATE_CLAIMED",
                                      (String)null,
                                      (Integer)null,
                                      (TimeZone)null);
   ```

   Cette opération renvoie un ensemble de résultats de requête contenant une liste des tâches réclamées par l'utilisateur connecté.
2. Met en suspens l'instance de tâche.

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  task.suspend(tkiid);
}
```

Cette action met en suspens l'instance de tâche spécifiée. L'instance de tâche est placée dans l'état Interrompu.

3. Reprise de l'instance de processus.

   ```
   task.resume( tkiid );
   ```

   Cette action remet l'instance de tâche dans l'état où elle se trouvait avant sa mise en suspens.

Analyse des résultats d'une tâche

Une tâche à effectuer (également appelée tâche de *participation* dans l'API) ou une tâche de collaboration (également appelée *tâche manuelle* dans l'API) fonctionne de manière asynchrone. Si un gestionnaire de réponses est indiqué lors du démarrage d'une tâche, le message de sortie est automatiquement retourné à la fin de celle-ci. Dans le cas contraire, le message doit être extrait explicitement.

task_context

Les résultats de la tâche ne sont stockés dans la base de données que si le modèle de tâche dont dérive l'instance de tâche ne spécifie pas une suppression automatique des instances de tâche dérivées.

task_procedure

Analysez les résultats de la tâche.
L'exemple illustre le contrôle du numéro d'ordre d'une tâche effectuée avec succès.

```
QueryResultSet result = task.query("DISTINCT TASK.TKIID",
                                   "TASK.NAME = 'CustomerOrder' AND
                                    TASK.STATE = TASK.STATE.STATE_FINISHED",
                                    (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  ClientObjectWrapper output = task.getOutputMessage(tkiid);
  DataObject myOutput = null;
  if ( output.getObject() != null && output.getObject() instanceof DataObject)
  {
     myOutput  = (DataObject)output.getObject();
     int order = myOutput.getInt("OrderNo");
  }
}
```

Arrêt d'une instance de tâche

Il s'avère parfois nécessaire pour quelqu'un disposant de droits d'administration d'arrêter une instance de tâche dans un état irrécupérable. Etant donné qu'une instance de tâche s'arrête instantanément, cette opération ne doit être exécutée que dans des situations exceptionnelles.

task_procedure

1. Procédez à l'extraction de l'instance de tâche devant être arrêtée.

   ```
   Task taskInstance = task.getTask(tkiid);
   ```

2. Arrêtez l'instance de tâche.

```
TKIID tkiid = taskInstance.getID();
task.terminate(tkiid);
```

L'instance de tâche est arrêtée aussitôt sans attendre les tâches en instance.

Suppression d'instances de tâche

Les instances de tâche ne sont automatiquement supprimées que lorsqu'elles sont terminées, à condition que cela soit spécifié dans le modèle de tâche associé dont dérivent les instances. Cet exemple montre comment supprimer toutes les instances de tâche qui sont terminées mais ne sont pas supprimées automatiquement.

task_procedure

1. Répertoriez les instances de tâche qui sont terminées.

```
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.STATE = TASK.STATE.STATE_FINISHED",
                 (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie un ensemble de résultats de requête qui répertorie les instances de tâche terminées.

2. Supprimez les instances de tâche terminées.

```
while (result.next() )
{
 TKIID tkiid = (TKIID) result.getOID(1);
 task.delete(tkiid);
}
```

Libération d'une tâche réclamée

Lorsqu'un propriétaire potentiel réclame une tâche, il lui incombe de mener la tâche à son terme. Toutefois, certaines tâches réclamées doivent être libérées pour afin qu'un autre propriétaire potentiel puisse la réclamer à son tour.

task_context

Il s'avère parfois nécessaire pour un utilisateur disposant de droits d'administration de libérer une tâche réclamée. Cette situation peut se produire, par exemple, lorsqu'une tâche doit être effectuée en l'absence du propriétaire de la tâche. Le propriétaire de la tâche peut également libérer une tâche réclamée.

task_procedure

1. Répertoriez les tâches réclamées possédées par une personne spécifique, par exemple, Smith.

```
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.STATE = TASK.STATE.STATE_CLAIMED AND
                 TASK.OWNER = 'Smith'",
                 (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie un ensemble de résultats de requête répertoriant les tâches réclamées par cette personne, Smith.

2. Libérez la tâche réclamée.

```
if (result.size() > 0)
{
  result.first();
  TKIID tkiid = (TKIID) result.getOID(1);
  task.cancelClaim(tkiid, true);
}
```

Cette opération renvoie la tâche à l'état prêt de manière à ce qu'elle puisse être réclamée par l'un des autres propriétaires éventuels. Toute donnée de sortie définie par le propriétaire d'origine est maintenue.

Gestion des tâches élémentaires

Durant la durée de vie d'une instance d'activité ou de tâche, l'ensemble des personnes associées à l'objet peut changer, par exemple, si une personne est en congé, si de nouvelles personnes sont engagées ou si la charge de travail doit être redistribuée. Pour autoriser ces modifications, vous devez développer des applications afin de créer, supprimer ou transférer les tâches élémentaires.

task_context

Une tâche élémentaire correspond à l'affectation d'un objet à un utilisateur ou à un groupe d'utilisateurs pour un motif particulier. Cet objet est généralement une instance d'activité manuelle, une instance de processus ou une instance de tâche. Les motifs sont dérivés du rôle conféré à l'utilisateur pour l'objet. Un objet peut comporter plusieurs éléments de travail étant donné qu'un utilisateur peut avoir différents rôles associés à l'objet, et qu'un élément de travail est créé pour chacun de ces rôles. Une instance de tâche à effectuer peut par exemple avoir un élément de travail administrateur, lecteur, éditeur et propriétaire en même temps.

Les actions pouvant être menées pour gérer les tâches élémentaires dépendent du rôle de l'utilisateur : par exemple, un administrateur peut créer, supprimer et transférer des tâches élémentaires, alors que le propriétaire de la tâche ne peut que transférer des tâches élémentaires.

task_procedure

- Créez une tâche élémentaire.

```
// query the task instance for which an additional
// administrator is to be specified
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  // create the work item
  task.createWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_ADMINISTRATOR,"Smith");
}
```

  Cette opération crée une tâche élémentaire pour l'utilisateur Smith qui a un rôle d'administration.

- Supprimez une tâche élémentaire.

```
// query the task instance for which a work item is to be deleted
QueryResultSet result = task.query("TASK.TKIID",
                                   "TASK.NAME='CustomerOrder'",
                                    (String)null, (Integer)null,
                                    (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  // delete the work item
  task.deleteWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_READER,"Smith");
}
```

  Cette opération supprime la tâche élémentaire pour l'utilisateur Smith qui a un rôle de lecteur.

- Transférez une tâche élémentaire.

```
// query the task that is to be rescheduled
QueryResultSet result =
     task.query("DISTINCT TASK.TKIID",
                "TASK.NAME='CustomerOrder' AND
                 TASK.STATE=TASK.STATE.STATE_READY AND
                 WORK_ITEM.REASON=WORK_ITEM.REASON.REASON_POTENTIAL_OWNER AND
                 WORK_ITEM.OWNER_ID='Miller'",
                 (String)null, (Integer)null, (TimeZone)null);
if (result.size() > 0)
{
  result.first();
  // transfer the work item from user Miller to user Smith
  // so that Smith can work on the task
  task.transferWorkItem((TKIID)(result.getOID(1)),
                          WorkItem.REASON_POTENTIAL_OWNER,"Miller","Smith");
}
```

  Cette opération transfère la tâche élémentaire à l'utilisateur Smith de manière à ce qu'il puisse travailler avec.

Création de modèles de tâche et d'instances de tâche à l'exécution

Un outil de modélisation, comme WebSphere Integration Developer, permet habituellement de compiler des modèles de tâche. Vous installez les modèles de tâche dans WebSphere Process Server et créez des instances à partir de ces modèles en utilisant, par exemple, Business Process Choreographer Explorer. Cependant, vous pouvez également créer des instances de tâche manuelle ou de participation lors de l'exécution.

task_context

Cette opération peut être nécessaire, par exemple, quand la définition de tâche n'est pas disponible lors du déploiement de l'application, quand les tâches d'une procédure ne sont pas encore connues ou quand une tâche est requise pour mener à bien une collaboration ad hoc dans un groupe.

Vous pouvez modéliser les tâches à effectuer ou les tâches de collaboration ad hoc en créant des instances de la classe com.ibm.task.api.TaskModel, et les utiliser pour créer un modèle de tâche réutilisable ou créer directement une instance de tâche à exécution unique. Pour créer une instance de la classe TaskModel, un ensemble de méthodes de fabrique est disponible dans la classe de fabrique com.ibm.task.api.ClientTaskFactory. La modélisation des tâches manuelles lors de l'exécution se base sur EMF (Eclipse Modeling Framework).

task_procedure

1. Créez un ensemble de ressources org.eclipse.emf.ecore.resource.ResourceSet à l'aide de la méthode de fabrique createResourceSet.
2. OptionalColonSymbol Si vous avez l'intention d'utiliser des types de message complexes, vous pouvez soit les définir à l'aide de org.eclipse.xsd.XSDFactory, que vous pouvez obtenir grâce à la méthode de fabrique getXSDFactory(), soit importer directement un schéma XML existant à l'aide de la méthode de fabrique loadXSDSchema.

   Pour rendre les types complexes disponibles au serveur WebSphere Process Server, déployez-les dans le cadre d'une application d'entreprise.
3. Créez ou importez une définition WSDL (Web Services Definition Language) du type javax.wsdl.Definition.

Vous pouvez créer une nouvelle définition WSDL à l'aide de la méthode createWSDLDefinition. Puis vous pouvez lui ajouter un type de port et une opération. Vous pouvez également importer directement une définition WSDL existante à l'aide de la méthode de fabrique loadWSDLDefinition.

4. Créez la définition de tâche à l'aide de la méthode de fabrique createTTask.

   Si vous voulez ajouter ou manipuler des éléments de tâche plus complexes, vous pouvez utiliser la classe com.ibm.wbit.tel.TaskFactory que vous vous pouvez récupérer à l'aide de la méthode de fabrique getTaskFactory.
5. Créez le modèle de tâche en utilisant la méthode de fabrique createTaskModel, puis envoyez-lui le regroupement de ressources que vous avez créé à l'étape 1 et qui rassemble tous les autres artefacts que vous avez créés depuis lors.
6. OptionalColonSymbol Validez le modèle à l'aide de la méthode TaskModel validate.

task_results

Utilisez l'une des méthodes create de l'API EJB Human Task Manager dont le paramètre **TaskModel** permet de créer un modèle de tâche réutilisable ou de créer directement une instance de tâche à exécution unique.

Tâches associées

«Création de tâches d'exécution utilisant des types Java simples»
Cet exemple crée une tâche d'exécution utilisant des types Java simples, comme un objet String, dans son interface.

«Création de tâches d'exécution utilisant des types complexes», à la page 490
Cet exemple crée une tâche d'exécution utilisant des types complexes dans son interface. Les types complexes sont déjà définis, c'est-à-dire que le système de fichiers local du client possède des fichiers XSD contenant la description des types complexes.

«Création de tâches d'exécution utilisant une interface existante», à la page 491
Cet exemple crée une tâche d'exécution utilisant une interface déjà définie, c'est-à-dire que le système de fichiers local possède un fichier contenant la description de l'interface.

«Création de tâches d'exécution utilisant une interface à partir d'une application d'appel», à la page 493
Cet exemple crée une tâche d'exécution utilisant une interface appartenant à l'application d'appel. Par exemple, une tâche d'exécution est créée dans un fragment de code Java d'un processus métier et utilise une interface à partir de l'application de processus.

Création de tâches d'exécution utilisant des types Java simples :

Cet exemple crée une tâche d'exécution utilisant des types Java simples, comme un objet String, dans son interface.

task_context

L'exemple s'exécute uniquement à l'intérieur du contexte de l'application d'entreprise appelante pour laquelle les ressources sont chargées.

task_procedure

1. Accédez à ClientTaskFactory et créez un ensemble de ressources contenant les définitions du nouveau modèle de tâche.

   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```

2. Créez la définition WSDL et ajoutez les descriptions des opérations.

```
// Création de l'interface WSDL
Definition definition = factory.createWSDLDefinition
         ( resourceSet, new QName( "http://www.ibm.com/task/test/", "test" ) );

// Création d'un type de port
PortType portType = factory.createPortType( definition, "doItPT" );

// Création d'une opération ; les messages d'entrée et de sortie sont de type Chaîne :
// aucun message d'erreur n'est spécifié
Operation operation = factory.createOperation
        ( definition, portType, "doIt",
          new QName( "http://www.w3.org/2001/XMLSchema", "string" ),
          new QName( "http://www.w3.org/2001/XMLSchema", "string" ),
          (Map)null );
```

3. Créez le modèle EMF de la nouvelle tâche manuelle.

   Si vous créez une instance de tâche, une date valid-from (UTCDate) n'est pas obligatoire.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   Cette étape initialise les propriétés du modèle de tâche avec des valeurs par défaut.

4. Modifiez les propriétés du modèle de tâche manuelle.

```
// Utilisation des méthodes du package the com.ibm.wbit.tel package, par exemple :
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// Extraction de la fabrique de tâches pour créer ou modifier
les éléments de tâches composites TaskFactory taskFactory
= factory.getTaskFactory();

// Spécification des paramètres d'escalade
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// Création de 'escalationReceiver' et ajout d'instruction
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// Création d'escalade et ajout de destinataire
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. Créez le modèle de tâche contenant toutes les définitions de ressources.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. Validez le modèle de tâche et corrigez les éventuels incidents de validation rencontrés.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. Créez l'instance ou le modèle de tâche d'exécution.

   L'interface HumanTaskManagerService permet de créer l'instance de tâche ou le modèle de tâche. Du fait que l'application utilise des types Java simples uniquement, il est inutile de spécifier un nom d'application.

   - Le fragment de code suivant crée une instance de tâche :

     ```
     atask.createTask( taskModel, (String)null, "HTM" );
     ```

   - Le fragment de code suivant crée un modèle de tâche :

```
task.createTaskTemplate( taskModel, (String)null );
```

task_results

Si une instance de tâche d'exécution est créée, elle peut à présent être démarrée. Si un modèle de tâche d'exécution est créé, vous pouvez à présent créer des instances de tâche à partir du modèle.

Création de tâches d'exécution utilisant des types complexes :

Cet exemple crée une tâche d'exécution utilisant des types complexes dans son interface. Les types complexes sont déjà définis, c'est-à-dire que le système de fichiers local du client possède des fichiers XSD contenant la description des types complexes.

task_context

L'exemple s'exécute uniquement à l'intérieur du contexte de l'application d'entreprise appelante pour laquelle les ressources sont chargées.

task_procedure

1. Accédez à ClientTaskFactory et créer un ensemble de ressources contenant les définitions du nouveau modèle de tâche.
   ```
   ClientTaskFactory factory = ClientTaskFactory.newInstance();
   ResourceSet resourceSet = factory.createResourceSet();
   ```
2. Ajoutez les définitions XSD de vos types complexes à l'ensemble de ressources pour les mettre à votre disposition lors de la définition d'opérations.

   Les fichiers sont relatifs à l'emplacement d'exécution du code.
   ```
   factory.loadXSDSchema( resourceSet, "InputBO.xsd" );
   factory.loadXSDSchema( resourceSet, "OutputBO.xsd" );
   ```
3. Créez la définition WSDL et ajoutez les descriptions des opérations.
   ```
   // Création de l'interface WSDL
   Definition definition = factory.createWSDLDefinition
            ( resourceSet, new QName( "http://www.ibm.com/task/test/", "test" ) );

   // Création d'un type de port
   PortType portType = factory.createPortType( definition, "doItPT" );

   // Création d'une opération ; le message d'entrée est un objet InputBO,
   // le message de sortie un objet OutputBO ;
   // aucun message d'erreur n'est spécifié
   Operation operation = factory.createOperation
           ( definition, portType, "doIt",
             new QName( "http://Input", "InputBO" ),
             new QName( "http://Output", "OutputBO" ),
             (Map)null );
   ```
4. Créez le modèle EMF de la nouvelle tâche manuelle.

   Si vous créez une instance de tâche, une date valid-from (UTCDate) n'est pas obligatoire.
   ```
   TTask humanTask = factory.createTTask( resourceSet,
                                          TTaskKinds.HTASK_LITERAL,
                                          "TestTask",
                                          new UTCDate( "2005-01-01T00:00:00" ),
                                          "http://www.ibm.com/task/test/",
                                          portType,
                                          operation );
   ```

Cette étape initialise les propriétés du modèle de tâche avec des valeurs par défaut.

5. Modifiez les propriétés du modèle de tâche manuelle.

```
// Utilisation des méthodes du package the com.ibm.wbit.tel package, par exemple :
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// Extraction de la fabrique de tâches pour créer ou modifier
les éléments de tâches composites
TaskFactory taskFactory = factory.getTaskFactory();

// Spécification des paramètres d'escalade
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// Création de 'escalationReceiver' et ajout d'instruction
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// Création d'escalade et ajout de destinataire
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

6. Créer le modèle de tâche contenant toutes les définitions de ressources.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

7. Validez le modèle de tâche et corrigez les éventuels incidents de validation rencontrés.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

8. Créez l'instance ou le modèle de tâche d'exécution.

   L'interface HumanTaskManagerService permet de créer l'instance de tâche ou le modèle de tâche. Vous devez fournir un nom d'application contenant les définitions de type de données pour les rendre accessibles. L'application doit également contenir une tâche ou un processus factice permettant son chargement par Business Process Choreographer.

   - Le fragment de code suivant crée une instance de tâche :

     ```
     task.createTask( taskModel, "BOapplication", "HTM" );
     ```

   - Le fragment de code suivant crée un modèle de tâche :

     ```
     task.createTaskTemplate( taskModel, "BOapplication" );
     ```

task_results

Si une instance de tâche d'exécution est créée, elle peut à présent être démarrée. Si un modèle de tâche d'exécution est créé, vous pouvez à présent créer des instances de tâche à partir du modèle.

Création de tâches d'exécution utilisant une interface existante :

Cet exemple crée une tâche d'exécution utilisant une interface déjà définie, c'est-à-dire que le système de fichiers local possède un fichier contenant la description de l'interface.

task_context

L'exemple s'exécute uniquement à l'intérieur du contexte de l'application d'entreprise appelante pour laquelle les ressources sont chargées.

task_procedure

1. Accédez à ClientTaskFactory et créez un ensemble de ressources contenant les définitions du nouveau modèle de tâche.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();
ResourceSet resourceSet = factory.createResourceSet();
```

2. Accédez à la définition WSDL et aux descriptions des opérations.

   La description d'interface est relative à l'emplacement d'exécution du code.

```
Definition definition = factory.loadWSDLDefinition(
                          resourceSet, "interface.wsdl"  );
PortType portType = definition.getPortType(
                     new QName( definition.getTargetNamespace(), "doItPT" ) );
Operation operation = portType.getOperation
                               ("doIt", (String)null, (String)null);
```

3. Créez le modèle EMF de la nouvelle tâche manuelle.

   Si vous créez une instance de tâche, une date valid-from (UTCDate) n'est pas obligatoire.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   Cette étape initialise les propriétés du modèle de tâche avec des valeurs par défaut.

4. Modifiez les propriétés du modèle de tâche manuelle.

```
// Utilisation des méthodes du package the com.ibm.wbit.tel package, par exemple :
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );

// Extraction de la fabrique de tâches pour créer ou modifier
les éléments de tâches composites
TaskFactory taskFactory = factory.getTaskFactory();

// Spécification des paramètres d'escalade
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// Création de 'escalationReceiver' et ajout d'instruction
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// Création d'escalade et ajout de destinataire
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. Créez le modèle de tâche contenant toutes les définitions de ressources.

```
TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
```

6. Validez le modèle de tâche et corrigez les éventuels incidents de validation rencontrés.

```
ValidationProblem[] validationProblems = taskModel.validate();
```

7. Créez l'instance ou le modèle de tâche d'exécution.

   L'interface HumanTaskManagerService permet de créer l'instance de tâche ou le modèle de tâche. Vous devez fournir un nom d'application contenant les définitions de type de données pour les rendre accessibles. L'application doit également contenir une tâche ou un processus factice permettant son chargement par Business Process Choreographer.

   - Le fragment de code suivant crée une instance de tâche :

```
task.createTask( taskModel, "BOapplication", "HTM" );
```

- Le fragment de code suivant crée un modèle de tâche :

```
task.createTaskTemplate( taskModel, "BOapplication" );
```

task_results

Si une instance de tâche d'exécution est créée, elle peut à présent être démarrée. Si un modèle de tâche d'exécution est créé, vous pouvez à présent créer des instances de tâche à partir du modèle.

Création de tâches d'exécution utilisant une interface à partir d'une application d'appel :

Cet exemple crée une tâche d'exécution utilisant une interface appartenant à l'application d'appel. Par exemple, une tâche d'exécution est créée dans un fragment de code Java d'un processus métier et utilise une interface à partir de l'application de processus.

task_context

L'exemple s'exécute uniquement à l'intérieur du contexte de l'application d'entreprise appelante pour laquelle les ressources sont chargées.

task_procedure

1. Accédez à ClientTaskFactory et créez un ensemble de ressources contenant les définitions du nouveau modèle de tâche.

```
ClientTaskFactory factory = ClientTaskFactory.newInstance();

// Spécification du chargeur de classe de contexte pour rechercher
les ressources suivantes
ResourceSet resourceSet = factory.createResourceSet
              ( Thread.currentThread().getContextClassLoader() );
```

2. Accédez à la définition WSDL et aux descriptions des opérations.

   Indiquez le chemin d'accès à l'intérieur du fichier JAR de package contenant.

```
Definition definition = factory.loadWSDLDefinition( resourceSet,
                           "com/ibm/workflow/metaflow/interface.wsdl" );
                      PortType portType = definition.getPortType(
                      new QName( definition.getTargetNamespace(), "doItPT" ) );
Operation operation = portType.getOperation
                                  ("doIt", (String)null, (String)null);
```

3. Créez le modèle EMF de la nouvelle tâche manuelle.

   Si vous créez une instance de tâche, une date valid-from (UTCDate) n'est pas obligatoire.

```
TTask humanTask = factory.createTTask( resourceSet,
                                       TTaskKinds.HTASK_LITERAL,
                                       "TestTask",
                                       new UTCDate( "2005-01-01T00:00:00" ),
                                       "http://www.ibm.com/task/test/",
                                       portType,
                                       operation );
```

   Cette étape initialise les propriétés du modèle de tâche avec des valeurs par défaut.

4. Modifiez les propriétés du modèle de tâche manuelle.

```
// Utilisation des méthodes du package the com.ibm.wbit.tel package, par exemple :
humanTask.setBusinessRelevance( TBoolean, YES_LITERAL );
```

```
// Extraction de la fabrique de tâches pour créer ou modifier
les éléments de tâches composites
TaskFactory taskFactory = factory.getTaskFactory();

// Spécification des paramètres d'escalade
TVerb verb = taskFactory.createTVerb();
verb.setName("John");

// Création de 'escalationReceiver' et ajout d'instruction
TEscalationReceiver escalationReceiver =
                    taskFactory.createTEscalationReceiver();
escalationReceiver.setVerb(verb);

// Création d'escalade et ajout de destinataire
TEscalation escalation = taskFactory.createTEscalation();
escalation.setEscalationReceiver(escalationReceiver);
```

5. Créez le modèle de tâche contenant toutes les définitions de ressources.

   ```
   TaskModel taskModel = ClientTaskFactory.createTaskModel( resourceSet );
   ```

6. Validez le modèle de tâche et corrigez les éventuels incidents de validation rencontrés.

   ```
   ValidationProblem[] validationProblems = taskModel.validate();
   ```

7. Créez l'instance ou le modèle de tâche d'exécution.

   L'interface HumanTaskManagerService permet de créer l'instance de tâche ou le modèle de tâche. Vous devez fournir un nom d'application contenant les définitions de type de données pour les rendre accessibles.

   - Le fragment de code suivant crée une instance de tâche :

     ```
     task.createTask( taskModel, "WorkflowApplication", "HTM" );
     ```

   - Le fragment de code suivant crée un modèle de tâche :

     ```
     task.createTaskTemplate( taskModel, "WorkflowApplication" );
     ```

task_results

Si une instance de tâche d'exécution est créée, elle peut à présent être démarrée. Si un modèle de tâche d'exécution est créé, vous pouvez à présent créer des instances de tâche à partir du modèle.

Interface HumanTaskManagerService

L'interface HumanTaskManagerService permet l'accès aux fonctions relatives aux tâches pouvant être appelées par des clients locaux ou distants.

Différentes méthodes peuvent être appelées selon l'état de la tâche et les droits d'accès de l'utilisateur de l'application contenant la méthode en question. Les méthodes principales de manipulation des objets de tâche sont répertoriées dans cette rubrique. Plus plus d'information sur ces méthodes et d'autres méthodes fournies par l'interface HumanTaskManagerService, consultez Javadoc dans le package com.ibm.task.api.

Modèles de tâches

Les méthodes suivantes sont disponibles pour les modèles de tâches.

Tableau 65. Méthodes API pour les modèles de tâches

Méthode	Description
getTaskTemplate	Extrait le modèle de tâche spécifié.
createTask	Crée une instance de tâche à partir du modèle de tâche spécifié.

Tableau 65. Méthodes API pour les modèles de tâches (suite)

Méthode	Description
createAndCallTask	Crée et exécute une instance de tâche à partir du modèle de tâche et attend le résultat de façon synchrone.
createAndStartTask	Crée et démarre une instance de tâche à partir du modèle de tâche spécifié.
createAndStartTaskAsSubtask	Crée et démarre une instance de tâche en tant que sous-tâche de la tâche spécifiée.
createInputMessage	Crée un message d'entré pour le modèle de tâche indiqué. Par exemple, crée un message pouvant servir à démarrer une tâche.
queryTaskTemplates	Extrait des modèles de tâche stockés dans la base de données.

Instances de tâches

Les méthodes suivantes sont disponibles pour les instances de tâches.

Tableau 66. Méthodes API pour les modèles de tâches

Méthode	Description
getTask	Extrait une instance de tâche ; l'instance de tâche peut se trouver dans n'importe quel état.
callTask	Démarre une tâche d'appel en mode synchrone.
startTask	Démarre une tâche qui a déjà été créée.
startTaskAsSubtask	Démarre une tâche en tant que sous-tâche de l'instance de tâche.
suspend	Interrompt la tâche de collaboration ou la tâche à effectuer.
resume	Reprend la tâche de collaboration ou la tâche à effectuer.
restart	Redémarre l'instance de tâche.
terminate	Arrête l'instance de tâche spécifiée. Si une tâche d'appel est arrêtée, cette action n'a aucun impact sur le service appelé.
delete	Supprime l'instance de tâche spécifiée.
claim	Réclame la tâche en vue de son traitement.
update	Met à jour l'instance de tâche.
complete	Termine l'instance de tâche.
completeWithFollowOnTask	Termine l'instance de tâche et démarre une tâche de suivi.
cancelClaim	Libère une instance de tâche réclamée afin de permettre son traitement par un autre propriétaire potentiel.
createWorkItem	Crée un élément de travail pour l'instance de tâche.

Tableau 66. Méthodes API pour les modèles de tâches (suite)

Méthode	Description
transferWorkItem	Transfère l'élément de travail à un propriétaire spécifié.
deleteWorkItem	Supprime l'élément de travail.

Escalades

Les méthodes suivantes sont disponibles pour les escalades.

Tableau 67. Méthodes API de gestion des escalades

Méthode	Description
getEscalation	Extrait l'instance d'escalade spécifiée.
triggerEscalation	Déclenche manuellement une escalade.

Propriétés personnalisées

Les tâches, les modèles de tâche et les escalades peuvent tous posséder des propriétés personnalisées. L'interface fournit une méthode get et une méthode set pour l'extraction et la définition de valeurs des propriétés personnalisées. Vous pouvez aussi associer les propriétés mentionnées aux instances de tâche et les en extraire. Le noms de propriétés personnalisées et des valeurs doivent être de type java.lang.String. Les méthodes suivantes sont adaptées aux tâches, modèles de tâche et escalades.

Tableau 68. Méthodes API pour les variables et les propriétés personnalisées

Méthode	Description
getCustomProperty	Extrait la propriété personnalisée mentionnée de l'instance de tâche spécifiée.
getCustomProperties	Extrait les propriétés personnalisées de l'instance de tâche spécifiée.
getCustomPropertyNames	Extrait les noms des propriétés personnalisées pour l'instance de tâche.
setCustomProperty	Stocke les valeurs spécifiques aux propriétés personnalisées correspondant à l'instance de tâche spécifiée.

Développement d'applications pour les processus métier et les tâches manuelles

La plupart des scénarios de processus métier nécessitent la participation de personnes. Par exemple, un processus métier nécessite une interaction manuelle lorsque le processus est démarré ou géré ou lorsque des activités manuelles sont effectuées. Pour supporter de tels scénarios, vous devez utiliser à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

task_context

Pour impliquer des personnes dans des scénarios de processus métier, vous pouvez inclure les types de tâche suivants dans le processus métier :

- Une tâche d'appel en ligne (également appelée *tâche de départ* dans l'API).

Vous pouvez fournir une tâche d'appel pour chaque activité de réception, pour chaque élément onMessage de l'activité de sélection et pour chaque élément onEvent du gestionnaire d'événements. Cette tâche peut alors contrôler les utilisateurs autorisés à démarrer un processus ou à communiquer avec une instance de processus en cours d'exécution.

- Une tâche d'administration.

  Vous pouvez fournir une tâche d'administration afin d'indiquer qui est autorisé à administrer le processus ou à effectuer des opérations d'administration sur les activité du processus qui ont échoué.

- Une tâche à effectuer (également appelée *tâche de participation* dans l'API).

  Les tâches à effectuer implémentent une activité manuelle. Ce type d'activité vous permet de faire participer des utilisateurs au processus.

Les activités manuelles du processus métier représentent les tâches à effectuer réalisées par les utilisateurs dans le scénario de processus métier. Pour réaliser de tels scénarios, vous pouvez utiliser à la fois l'API de Business Flow Manager et l'API de Human Task Manager.

- Le processus métier est le conteneur de toutes les activités appartenant au processus, y compris les activités manuelles qui sont représentées par les tâches à effectuer. Lorsqu'une instance de processus est créée, un ID objet unique (PIID) lui est affecté.
- Lorsqu'une activité manuelle est activée au cours de l'exécution de l'instance de processus, une instance d'activité est créée, qui est identifiée par son ID objet (AIID) unique. En même temps, une instance de tâche à effectuer en ligne est également créée, qui est identifiée par son ID objet (TKIID). La relation entre l'activité manuelle et l'instance de tâche est créée par le biais des ID objet :
  - L'ID tâche à effectuer de l'instance d'activité est défini en fonction du TKIID de la tâche à effectuer associée.
  - L'ID de contexte de confinement de l'instance de tâche est défini en fonction de l'instance de processus qui contient l'instance d'activité associée.
  - L'ID de contexte parent de l'instance de tâche est défini en fonction de l'AIID de l'instance d'activité associée.
- Les cycles de vie de toutes les instances de tâche à effectuer en ligne sont gérés par l'instance de processus. Lorsque l'instance de processus est supprimée, les instances de tâches le sont également. Par exemple, toutes les tâches dont l'ID de contexte de confinement est défini en fonction du PIID de l'instance de processus sont automatiquement supprimées.

Tâches associées

«Déterminer les modèles de processus ou les activités pouvant être démarrés»
Un processus métier peut être démarré en appelant les méthodes call, initiate ou sendMessage de l'API de Business Flow Manager. Si le processus n'a qu'une seule activité de démarrage, vous pouvez utiliser la signature de méthode dont le paramètre doit être un nom de modèle de processus. Si le processus comporte plusieurs activités de démarrage, vous devez identifier l'activité de démarrage de manière explicite.

«Traitement par une seule personne d'un flux de travaux contenant des tâches manuelles», à la page 500
Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Cet exemple montre comment implémenter un flux de travaux exécutés par une seule personne à l'aide d'un flux de pages côté serveur. Les API de Business Flow Manager et Human Task Manager sont toutes les deux utilisées pour traiter le flux de travaux.

Déterminer les modèles de processus ou les activités pouvant être démarrés

Un processus métier peut être démarré en appelant les méthodes call, initiate ou sendMessage de l'API de Business Flow Manager. Si le processus n'a qu'une seule activité de démarrage, vous pouvez utiliser la signature de méthode dont le paramètre doit être un nom de modèle de processus. Si le processus comporte plusieurs activités de démarrage, vous devez identifier l'activité de démarrage de manière explicite.

task_context

Lorsqu'un processus métier est modélisé, le modélisateur peut décider que seul un sous-ensemble d'utilisateurs est autorisé à créer une instance de processus à partir du modèle de processus. Ceci est effectué en associant une tâche d'appel en ligne à une activité de démarrage du processus, puis en précisant les restrictions d'autorisation appliquées à cette tâche. Seuls les utilisateurs qui sont des démarreurs ou des administrateurs potentiels de la tâche sont autorisés à créer une instance de la tâche, et par conséquent, une instance du modèle de processus.

Si aucune tâche d'appel en ligne n'est associée à l'activité de démarrage, ou si les restrictions d'autorisation ne sont pas indiquées pour la tâche, tous les utilisateurs peuvent créer une instance de processus à l'aide de l'activité de démarrage.

Un processus peut avoir plusieurs activités de démarrage, chacune avec différentes requêtes d'utilisateurs pour des démarreurs ou des administrateurs potentiels. Cela signifie qu'un utilisateur peut être autorisé à démarrer un processus avec l'activité A, mais pas avec l'activité B.

task_procedure

1. Utilisez l'API de Business Flow Manager pour créer la liste des versions courantes des modèles de processus qui sont à l'état démarré.

   Tip : La méthode queryProcessTemplates exclut uniquement les modèles de processus qui font partie des applications n'ayant pas encore démarré. Par conséquent, si vous utilisez cette méthode sans filtrer les résultats, vous obtiendrez toutes les versions des modèles de processus indépendamment de leur état.

```
// current timestamp in UTC format, converted to yyyy-mm-ddThh:mm:ss
String now = (new UTCDate()).toXsdString();
String whereClause = "PROCESS_TEMPLATE.STATE =
                      PROCESS_TEMPLATE.STATE.STATE_STARTED AND
                      PROCESS_TEMPLATE.VALID_FROM =
                      (SELECT MAX(VALID_FROM) FROM PROCESS_TEMPLATE
                       WHERE NAME=PROCESS_TEMPLATE.NAME AND
                       VALID_FROM <= TS('" + now + "'))";

ProcessTemplateData[] processTemplates = process.queryProcessTemplates
                                        ( whereClause,
                                          "PROCESS_TEMPLATE.NAME",
                                          (Entier)null, (FuseauHoraire)null);
```

Les résultats sont triés par nom de modèle de processus.

2. Créez la liste des modèles de processus et celle des activités de démarrage pour lesquelles l'utilisateur est autorisé.

   La liste des modèles de processus contient les modèles de processus ayant une activité de démarrage unique. Soit ces activités sont non protégées, soit l'utilisateur connecté est autorisé à les démarrer. Sinon, vous pouvez regrouper les modèles de processus qui peuvent être démarrés par au moins une activité de démarrage.

   Tip : Un administrateur de processus peut également créer une instance de processus. Toutefois, si Business Flow Manager utilise le mode d'autorisation pour l'administration alternative du processus, qui restreint l'administration des processus aux administrateurs système, seuls les utilisateurs du rôle BPESystemAdministrator peuvent effectuer cette action. Par conséquent, pour obtenir la liste complète des modèles, vous devez également vérifier si l'utilisateur est connecté en tant qu'administrateur.

   ```
   List authorizedProcessTemplates = new ArrayList();
   List authorizedActivityServiceTemplates = new ArrayList();
   ```

3. Déterminez les activités de démarrage pour chacun des modèles de processus.

   ```
   for( int i=0; i<processTemplates.length; i++ )
   {
     ProcessTemplateData template = processTemplates[i];
     ActivityServiceTemplateData[] startActivities =
                   process.getStartActivities(template.getID());
   ```

4. Pour chaque activité de démarrage, récupérez l'ID du modèle de tâche d'appel en ligne associé.

   ```
   for( int j=0; j<startActivities.length; j++ )
   {
     ActivityServiceTemplateData activity = startActivities[j];
     TKTID tktid = activity.getTaskTemplateID();
   ```

   a. Si un modèle de tâche d'appel n'existe pas, cela signifie que le modèle de processus n'est pas sécurisé par cette activité de démarrage.

      Dans pareil cas, tout utilisateur peut créer une instance de processus à l'aide de cette activité de démarrage.

      ```
      boolean isAuthorized = false;
             if ( tktid == null )
          {
            isAuthorized = true;
            authorizedActivityServiceTemplates.add(activity);
          }
      ```

   b. Si un modèle de tâche d'appel existe, utilisez l'API de Human Task Manager pour vérifier les autorisations dont dispose l'utilisateur connecté.

Dans l'exemple, l'utilisateur connecté s'appelle Smith. Il est impératif que l'utilisateur connecté soit un démarreur potentiel de la tâche d'appel ou un administrateur.

```
if ( tktid != null )
    {
      isAuthorized =
        task.isUserInRole
            (tkid, "Smith", WorkItem.REASON_POTENTIAL_STARTER) ||
        task.isUserInRole(tktid, "Smith", WorkItem.REASON_ADMINISTRATOR);

      if ( isAuthorized )
      {
        authorizedActivityServiceTemplates.add(activity);
      }
    }
```

Si l'utilisateur correspond au rôle indiqué ou si les critères d'affectation des utilisateurs pour ce rôle ne sont pas définis, la méthode isUserInRole renvoie la valeur `true`.

5. Vérifiez s'il est possible de démarrer le processus à l'aide du nom du modèle de processus uniquement.

```
if ( isAuthorized && startActivities.length == 1 )
    {
      authorizedProcessTemplates.add(template);
    }
```

6. Arrêtez les boucles.

```
  } // end of loop for each activity service template
}   // end of loop for each process template
```

Traitement par une seule personne d'un flux de travaux contenant des tâches manuelles

Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Cet exemple montre comment implémenter un flux de travaux exécutés par une seule personne à l'aide d'un flux de pages côté serveur. Les API de Business Flow Manager et Human Task Manager sont toutes les deux utilisées pour traiter le flux de travaux.

task_context

Un flux de travaux exécuté par une seule personne est également appelé *flux de pages* ou *flux d'écrans*. Il existe deux types de flux de pages :

- Les flux de pages côté client, où la navigation entre les différentes pages est effectuée à l'aide de la technologie côté client (par exemple, un formulaire Lotus Forms sur plusieurs pages).
- Les flux de pages côté serveur sont réalisés à l'aide d'un processus métier et d'un ensemble de tâches manuelles modélisées de sorte que les tâches suivantes soient affectées à la même personne.

Les flux de pages côté serveur sont plus puissants que les flux de pages côté client, mais ils utilisent davantage de ressources serveur en vue de leur traitement. Par conséquent, il est recommandé de n'utiliser ce type de flux de travaux que dans les cas suivants :

- Vous devez appeler des services entre les étapes effectuées dans une interface utilisateur (par exemple, pour extraire ou mettre à jour des données).
- Certaines de vos exigences d'audit requièrent l'écriture d'événements CEI après interaction avec une interface utilisateur.

Dans une librairie en ligne, l'acheteur accomplit une série d'actions afin de commander un ouvrage. Cette séquence d'actions peut être implémentée comme une série d'activités manuelles (tâches à accomplir). Si l'acheteur décide de commander plusieurs livres, cela équivaut à réclamer l'activité manuelle suivante. Les informations sur la séquence de tâches sont gérées par le Business Flow Manager, alors que les tâches elles-mêmes sont gérées par le Human Task Manager.

Comparez cet exemple avec celui qui utilise uniquement l'API de Business Flow Manager.

task_procedure

1. Utilisez l'API de Business Flow Manager pour accéder à l'instance de processus que vous voulez traiter.

   Dans cet exemple, il s'agit d'une instance du processus CustomerOrder.

   ```
   ProcessInstanceData processInstance =
      process.getProcessInstance("CustomerOrder");
   String piid = processInstance.getID().toString();
   ```

2. Utilisez l'API de Human Task Manager pour interroger les tâches à effectuer prêtes (de type tâche de participation) qui font partie de l'instance de processus indiquée.

   Utilisez l'ID de contexte de confinement de la tâche pour spécifier l'instance du processus de confinement. Pour un flux de travaux exécuté par une seule personne, la requête renvoie la tâche à effectuer qui est associée à la première activité manuelle dans la séquence d'activités manuelles.

   ```
   //
    // Query the list of to-do tasks that can be claimed by the logged-on user
    // for the specified process instance
   //
   QueryResultSet result =
        task.query("DISTINCT TASK.TKIID",
                   "TASK.CONTAINMENT_CTX_ID = ID('" + piid + "') AND
                    TASK.STATE = TASK.STATE.STATE_READY AND
                    TASK.KIND = TASK.KIND.KIND_PARTICIPATING AND
                    WORK_ITEM.REASON=WORK_ITEM.REASON.REASON_POTENTIAL_OWNER",
                   (String)null, (Integer)null, (TimeZone)null);
   ```

3. Réclamez la tâche à effectuer qui est renvoyée.

   ```
   if (result.size() > 0)
   {
    result.first();
    TKIID tkiid = (TKIID) result.getOID(1);
    ClientObjectWrapper input = task.claim(tkiid);
    DataObject activityInput = null ;
     if ( input.getObject()!= null && input.getObject()
   instanceof DataObject )
     {
       taskInput = (DataObject)input.getObject();
       // read the values
       ...
     }
   }
   ```

   Une fois la tâche réclamée, le message d'entrée de la tâche est renvoyé.

4. Déterminez l'activité manuelle qui est associée à la tâche à effectuer.

   Pour établir une corrélation entre les activités et les tâches correspondantes, vous pouvez utiliser l'une des méthodes suivantes.

   - La méthode task.getActivityID :

```
AIID aiid = task.getActivityID(tkiid);
```

- L'ID de contexte parent qui fait partie de l'objet tâche :

```
AIID aiid = null;
Task taskInstance = task.getTask(tkiid);

OID oid = taskInstance.getParentContextID();
if ( oid != null and oid instanceof AIID )
{
  aiid = (AIID)oid;
}
```

5. Lorsque vous avez terminé de traiter la tâche, utilisez l'API de Business Flow Manager pour terminer la tâche ainsi que l'activité manuelle qui lui est associée, puis réclamez l'activité manuelle suivante dans l'instance de processus.

   Pour terminer l'activité manuelle, un message de sortie est transmis. Lorsque vous créez le message de sortie, vous devez spécifier le nom de son type de message de manière à ce qu'il contienne la définition du message.

```
ActivityInstanceData activity = process.getActivityInstance(aiid);
ClientObjectWrapper output =
      process.createMessage(aiid, activity.getOutputMessageTypeName());
DataObject myMessage = null ;
if ( output.getObject()!= null && output.getObject() instanceof DataObject )
{
  myMessage = (DataObject)output.getObject();
  //set the parts in your message, for example, an order number
  myMessage.setInt("OrderNo", 4711);
}

//complete the human task activity and its associated to-do task,
// and claim the next human task activity
CompleteAndClaimSuccessorResult successor =
   process.completeAndClaimSuccessor(aiid, output);
```

   Cette opération définit un message de sortie contenant le numéro de commande et réclame l'activité manuelle suivante de la séquence. Si `AutoClaim` est défini pour les activités de succession et que plusieurs chemins d'accès peuvent être utilisés, toutes les activités de succession sont réclamées et une activité aléatoire est renvoyée en tant qu'activité suivante. Si aucune activité de succession supplémentaire ne peut être affectée à cet utilisateur, la valeur `Null` est renvoyée.

   Si le processus contient des chemins parallèles pouvant être suivis, que ces chemins contiennent des activités manuelles et que l'utilisateur connecté est le propriétaire potentiel de plusieurs de ces activités, une activité aléatoire est automatiquement réclamée et renvoyée comme activité suivante.

6. Exécutez l'activité manuelle suivante.

```
ClientObjectWrapper nextInput = successor.getInputMessage();
if ( nextInput.getObject()!=
               null && nextInput.getObject() instanceof DataObject )
{
  activityInput = (DataObject)input.getObject();
  // read the values
  ...
}

aiid = successor.getAIID();
```

7. Passez à l'étape 5 afin de terminer l'activité manuelle et de récupérer l'activité manuelle suivante.

Tâches associées

«Traitement d'un flux de travaux par une seule personne», à la page 462
Certains flux de travaux sont exécutés par une seule personne, par exemple une commande d'ouvrages sur une librairie en ligne. Ce type de flux de travaux ne comporte pas de chemins d'accès parallèles. Les API initiateAndClaimFirst et completeAndClaimSuccessor prennent en charge le traitement de ce type de flux de travaux. Cet exemple illustre l'implémentation d'un flux de travaux exécutés par une seule personne à l'aide d'un flux de pages côté client.

Gestion des exceptions et des erreurs

Un processus BPEL peut rencontrer une erreur à différents points du processus.

task_context

Les erreurs BPEL (Business Process Execution Language) proviennent des éléments suivants :

- Appels de service Web (erreurs WSDL (Web Services Description Language))
- Activités d'émission
- Erreurs standard BPEL reconnues par Business Process Choreographer

Il existe des mécanismes pour gérer ces erreurs : Pour résoudre les erreurs liées à une instance de processus, utilisez l'un des mécanismes suivants :

- Transférez le contrôle aux gestionnaires d'erreur correspondants
- Effectuez une compensation du travail précédent du processus
- Arrêtez le processus afin de laisser quelqu'un d'autre remédier à la situation (forcer la nouvelle tentative, forcer à terminer)

Un processus BPEL peut également renvoyer des erreurs à l'appelant d'une opération fournie par le processus. Vous pouvez modéliser l'erreur dans le processus sous forme d'activité de réponse avec un nom d'erreur et des données d'erreur. Ces erreurs sont renvoyées à l'appelant API sous forme d'exceptions vérifiées.

Si un processus BPEL ne gère pas d'erreurs BPEL ou si une exception API survient, une exception d'exécution est renvoyée à l'appelant de l'API. Par exemple, une exception API est lancée lorsque le modèle de processus à partir duquel une instance doit être créée n'existe pas.

La gestion des erreurs et des exceptions est décrite dans les tâches suivantes.

Tâches associées

«Gestion des exceptions de l'API EJB de Business Process Choreographer»
Si une méthode de l'interface BusinessFlowManagerService ou HumanTaskManagerService ne se termine pas correctement, une exception est générée, indiquant la cause de l'erreur. Vous pouvez gérer cette exception de manière spécifique pour guider l'appelant.

«Vérification de l'erreur définie pour une activité de tâche manuelle»
Lorsqu'une activité de tâche manuelle est traitée, elle peut s'exécuter correctement. Dans ce cas, vous pouvez transmettre un message de sortie. Si l'activité de tâche manuelle ne se termine pas correctement, vous pouvez transmettre un message d'erreur.

«Vérification d'une erreur survenue lors d'une activité d'appel arrêtée», à la page 505
Dans un processus conçu de façon appropriée, les exceptions et les erreurs sont généralement gérées par des gestionnaires d'erreur. Vous pouvez extraire les informations relatives à l'exception ou à l'erreur qui s'est produite pour une activité d'appel provenant de l'instance d'activité.

«Vérification de l'erreur ou de l'exception non gérée survenue lors de l'échec d'une instance de processus», à la page 506
Dans un processus conçu de façon appropriée, les exceptions et les erreurs sont généralement gérées par un gestionnaire d'erreur. Si le processus implémente une opération bi-directionnelle, vous pouvez extraire des informations sur une erreur ou une exception gérée à partir de la propriété du nom de l'erreur de l'objet de l'instance de processus. Pour les erreurs, vous pouvez également extraire le message d'erreur correspondant à l'aide de l'API getFaultMessage.

Gestion des exceptions de l'API EJB de Business Process Choreographer

Si une méthode de l'interface BusinessFlowManagerService ou HumanTaskManagerService ne se termine pas correctement, une exception est générée, indiquant la cause de l'erreur. Vous pouvez gérer cette exception de manière spécifique pour guider l'appelant.

task_context

Cependant, il est de coutume de gérer uniquement un sous-ensemble des exceptions de manière spécifique et de fournir un guide général pour les autres exceptions potentielles. Toutes les exceptions spécifiques héritent d'une classe générique ProcessException ou TaskException. Interceptez les exceptions génériques avec une instruction finale `catch(ProcessException)` ou `catch(TaskException)`. Cette instruction permet de veiller à la compatibilité ascendante de votre programme d'application car elle prend en compte toutes les autres exceptions qui peuvent survenir.

Vérification de l'erreur définie pour une activité de tâche manuelle

Lorsqu'une activité de tâche manuelle est traitée, elle peut s'exécuter correctement. Dans ce cas, vous pouvez transmettre un message de sortie. Si l'activité de tâche manuelle ne se termine pas correctement, vous pouvez transmettre un message d'erreur.

task_context

Vous pouvez lire le message d'erreur pour déterminer la cause de l'erreur.

task_procedure

1. Répertoriez les activités de tâche se trouvant à l'état d'échec ou arrêté.

```
QueryResultSet result =
     process.query("ACTIVITY.AIID",
                   "(ACTIVITY.STATE = ACTIVITY.STATE.STATE_FAILED OR
                     ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED) AND
                     ACTIVITY.KIND=ACTIVITY.KIND.KIND_STAFF",
                     (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie un ensemble de résultats de requête contenant des activités en échec ou arrêtées.

2. Lisez le nom de l'erreur.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);
  ClientObjectWrapper faultMessage = process.getFaultMessage(aiid);
  DataObject fault = null ;
  if ( faultMessage.getObject() != null && faultMessage.getObject() instanceof DataObject )
  {
    fault = (DataObject)faultMessage.getObject();
    Type type = fault.getType();
    String name = type.getName();
    String uri = type.getURI();
  }
}
```

   Cela renvoie le nom de l'erreur. Vous pouvez aussi analyser l'exception non prise en charge d'une activité arrêtée au lieu d'extraire le nom de l'erreur.

Vérification d'une erreur survenue lors d'une activité d'appel arrêtée

Dans un processus conçu de façon appropriée, les exceptions et les erreurs sont généralement gérées par des gestionnaires d'erreur. Vous pouvez extraire les informations relatives à l'exception ou à l'erreur qui s'est produite pour une activité d'appel provenant de l'instance d'activité.

task_context

Si une activité entraîne une erreur, le type d'erreur détermine les actions que vous pouvez effectuer pour réparer l'activité.

task_procedure

1. Répertoriez les activités manuelles qui sont en état arrêté.

```
QueryResultSet result =
     process.query("ACTIVITY.AIID",
                   "ACTIVITY.STATE = ACTIVITY.STATE.STATE_STOPPED AND
                    ACTIVITY.KIND=ACTIVITY.KIND.KIND_INVOKE",
                    (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie un ensemble de résultats de requête contenant des activités d'appel arrêtées.

2. Lisez le nom de l'erreur.

```
if (result.size() > 0)
{
  result.first();
  AIID aiid = (AIID) result.getOID(1);
  ActivityInstanceData activity = process.getActivityInstance(aiid);

  ProcessException excp = activity.getUnhandledException();
```

```
  if ( excp instanceof ApplicationFaultException )
  {
   ApplicationFaultException fault = (ApplicationFaultException)excp;
   String faultName = fault.getFaultName();
  }
}
```

Vérification de l'erreur ou de l'exception non gérée survenue lors de l'échec d'une instance de processus

Dans un processus conçu de façon appropriée, les exceptions et les erreurs sont généralement gérées par un gestionnaire d'erreur. Si le processus implémente une opération bi-directionnelle, vous pouvez extraire des informations sur une erreur ou une exception gérée à partir de la propriété du nom de l'erreur de l'objet de l'instance de processus. Pour les erreurs, vous pouvez également extraire le message d'erreur correspondant à l'aide de l'API getFaultMessage.

task_context

Si une instance de processus échoue parce qu'une exception n'est pas gérée par l'un des gestionnaire d'erreur, vous pouvez extraire des informations sur l'exception non gérée à partir de l'objet de l'instance de processus. En revanche, si une erreur est interceptée par un gestionnaire d'erreur, les informations sur l'erreur ne sont pas disponibles. Vous pouvez, cependant, extraire le message et le nom de l'erreur et les renvoyer à l'appelant à l'aide de l'exception FaultReplyException.

task_procedure

1. Répertoriez les instances de processus présentant l'état Echoué.

```
QueryResultSet result =
     process.query("PROCESS_INSTANCE.PIID",
                   "PROCESS_INSTANCE.STATE =
                       PROCESS_INSTANCE.STATE.STATE_FAILED",
                    (String)null, (Integer)null, (TimeZone)null);
```

   Cette opération renvoie un ensemble de résultats de requête contenant les instances de processus ayant échoué.

2. Prenez connaissances des informations concernant l'exception non gérée.

```
if (result.size() > 0)
{
  result.first();
  PIID piid = (PIID) result.getOID(1);
  ProcessInstanceData pInstance = process.getProcessInstance(piid);

  ProcessException excp = pInstance.getUnhandledException();
  if ( excp instanceof RuntimeFaultException )
  {
   RuntimeFaultException xcp = (RuntimeFaultException)excp;
   Throwable cause = xcp.getRootCause();
  }
  else if ( excp instanceof StandardFaultException )
  {
   StandardFaultException xcp = (StandardFaultException)excp;
   String faultName = xcp.getFaultName();
  }
  else if ( excp instanceof ApplicationFaultException )
  {
   ApplicationFaultException xcp = (ApplicationFaultException)excp;
   String faultName = xcp.getFaultName();
  }
}
```

task_results

Utilisez ces informations pour rechercher le nom de l'erreur ou la cause principale du problème.

Développement d'applications client de services Web pour des processus métier et des tâches manuelles

Vous pouvez développer des applications client accédant à des applications de processus métier et de tâches manuelles via des API de services Web Business Process Choreographer. Le processus de développement d'applications client comprend un certain nombre d'étapes obligatoires et facultatives, notamment la génération d'un proxy de service Web et l'ajout de règles de sécurité et de transaction dans l'application client.

task_context

A partir de la version 7, l'API des services Web JAX-WS remplace l'API des services Web JAX-RPC Business Process Choreographer dans la version 6 (première publication dans l'édition 6.0.2). L'API de services Web JAX-RPC Business Process Choreographer sera obsolète, ainsi les nouvelles applications client de services Web doivent être implémentées à l'aide de l'API JAX-WS.

Remarque : L'API Java Message Service (JMS) de Business Process Choreographer utilise toujours les définitions de schéma WSDL et XML pour la version 6.

Vous pouvez développer des applications client dans n'importe quel environnement client de services Web. La procédure ci-dessous donne un aperçu des actions à entreprendre pour développer une telle application.

task_procedure

1. Décidez quelle API de services Web votre application client doit utiliser : l'API de Business Flow Manager, l'API de Human Task Manager ou les deux.
2. Exportez les fichiers nécessaires depuis l'environnement de WebSphere Process Server.
3. Dans votre environnement de développement d'applications client, générez un *proxy de services Web* à l'aide des artefacts exportés.
4. Développez le code de votre application client.
5. Ajoutez les mécanismes de sécurité ou les règles de transaction nécessaires à votre application client.

Concepts associés

«Composants de service Web et séquence de contrôle»
Dans les applications de services Web, un certain nombre de composants côté client et côté serveur font partie de la séquence de contrôle qui représente une requête et une réponse de service Web.

«Exigences liées à l'API de service Web pour les processus métier et les tâches manuelles», à la page 509
Les processus métier et les tâches manuelles développés au moyen de WebSphere Integration Developer pour être exécutés dans l'application Business Process Choreographer doivent être conformes à des règles spécifiques afin d'être accessibles via les API de services Web.

«Développement d'applications client dans l'environnement de services Web Java», à la page 516
Vous pouvez utiliser tout environnement de développement Java compatible avec les services Web Java en vue de développer des applications client pour les API de services Web Business Process Choreographer.

Tâches associées

«Publication et exportation d'artefacts depuis l'environnement de serveur pour les applications client de services Web», à la page 511
Pour pouvoir développer des applications client en vue d'accéder aux API de services Web Business Process Choreographer, vous devez publier et exporter un certain nombre d'artefacts à partir de l'environnement de serveur WebSphere.

«Ajout de sécurité», à la page 520
Le service Web de Business Process Choreographer nécessite que vous configuriez votre application client pour un mécanisme d'authentification.

«Ajout d'un support de transaction», à la page 521
Les applications client de service Web peuvent être configurées pour permettre au traitement de la requête côté serveur de participer à la transaction client, en transmettant un contexte d'application client en tant que requête de service. Ce support de transaction atomique est défini dans la spécification Web Services-Atomic Transaction (WS-AT).

Référence associée

«API des services Web Business Process Choreographer JAX-WS», à la page 510
A partir de la version 7, l'API des services Web JAX-WS remplace l'API des services Web JAX-RPC Business Process Choreographer dans la version 6 (publiée en premier dans l'édition 6.0.2). Deux interfaces de services Web Business Process Choreographer sont fournies, chacune avec leurs propres artefacts de fichier et espaces de noms de définition XML.

«API des services Web de Business Process Choreographer : valeurs standard», à la page 510
Utilisez les liens suivants pour retrouver les informations supplémentaires relatives aux valeurs standard applicables aux applications Web. Ces informations se trouvent sur des sites Internet non IBM dont le contenu technique est contrôlé par les détenteurs du site.

Composants de service Web et séquence de contrôle

Dans les applications de services Web, un certain nombre de composants côté client et côté serveur font partie de la séquence de contrôle qui représente une requête et une réponse de service Web.

Une séquence de contrôle typique se présente comme suit.

1. Côté client :

a. Une application client (fournie par l'utilisateur) émet une requête de service Web.
b. Un proxy de service Web (également fourni par l'utilisateur, mais pouvant être généré automatiquement à l'aide d'utilisateurs côté client) encapsule la demande de service dans une enveloppe de requête SOAP et fait suivre la demande à une adresse URL définie en tant que noeud final de service Web.

2. Le réseau transmet la requête au noeud final de service Web via le protocole HTTP ou HTTPS.
3. Côté serveur :
   a. L'API de service Web générique reçoit la requête et la décode.
   b. La requête est soit gérée directement par les composants génériques Business Flow Manager ou Human Task Manager, soit transmise au processus métier ou à la tâche manuelle spécifiés.
   c. Les données renvoyées sont encapsulées dans une enveloppe de réponse SOAP.
4. Le réseau transmet la réponse à l'environnement côté-client via le protocole HTTP ou HTTPS.
5. De retour côté client :
   a. L'infrastructure de développement côté client décode l'enveloppe de réponse SOAP.
   b. Le proxy de service Web extrait les données de la réponse SOAP et les transmet à l'application client.
   c. L'application client traite les données renvoyées selon les nécessités.

Exemple

La procédure suivante peut convenir pour une application client qui accède à l'API du service Web Human Task Manager afin de traiter une tâche à effectuer :

1. L'application client envoie un appel de service Web query à WebSphere Process Server demandant la liste des tâches à effectuer auxquelles devra participer un utilisateur.
2. WebSphere Process Server renvoie la liste des tâches à effectuer.
3. L'application client émet alors un appel de service Web claim pour réclamer une des tâches à effectuer.
4. WebSphere Process Server renvoie le message d'entrée pour la tâche.
5. L'application client envoie un appel de service Web complete pour achever la tâche par un message de sortie ou d'erreur.

Exigences liées à l'API de service Web pour les processus métier et les tâches manuelles

Les processus métier et les tâches manuelles développés au moyen de WebSphere Integration Developer pour être exécutés dans l'application Business Process Choreographer doivent être conformes à des règles spécifiques afin d'être accessibles via les API de services Web.

Les exigences sont les suivantes :

- Les interfaces des processus métier et des tâches manuelles doivent être définies à l'aide du style "document/literal wrapped" défini dans l'API Java pour la spécification de services Web XML (JAX-WS 2.0). Il s'agit du style par défaut défini pour l'ensemble des processus métier et des tâches manuelles développés avec WebSphere Integration Developer.

- N'utilisez pas l'attribut maxOccurs dans les éléments de paramètre des opérations, ou assurez-vous que la valeur de cet attribut est définie sur la valeur par défaut, `maxOccurs="1"`.
- Les messages d'erreur exposés par les processus métier et les tâches manuelles pour les opérations de service Web doivent comporter une partie message WSDL avec un élément de schéma XML. Par exemple :

```
<wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
```

Information associée

Page de téléchargement de Java API for XML-based Web Services (JAX-WS 2.0)

Quel style de langage WSDL dois-je utiliser ?

API des services Web Business Process Choreographer JAX-WS

A partir de la version 7, l'API des services Web JAX-WS remplace l'API des services Web JAX-RPC Business Process Choreographer dans la version 6 (publiée en premier dans l'édition 6.0.2). Deux interfaces de services Web Business Process Choreographer sont fournies, chacune avec leurs propres artefacts de fichier et espaces de noms de définition XML.

Le tableau suivant présente les artefacts de fichier et les espaces de noms de définition XML pour les services Web JAX-WS.

Tableau 69. Artefacts de fichier et espaces de noms de définition XML pour les services Web JAX-WS

Interface des services Web de Business Process Choreographer	Artefact de fichier des services Web JAX-WS	Espaces de noms XML des services Web JAX-WS
Service Web de Business Flow Manager	BFMJAXWSService.wsdl	http://www.ibm.com/xmlns/prod/websphere/business-process/services/7.0/Binding
Interface de services Web de Business Flow Manager	BFMJAXWSInterface.wsdl	http://www.ibm.com/xmlns/prod/websphere/business-process/services/7.0
Types de données des services Web de Business Flow Manager	BFMDataTypes.xsd	http://www.ibm.com/xmlns/prod/websphere/business-process/types/7.0
Interface de service Web de rappel de Business Flow Manager	BFMJAXWSCallbackService.wsdl	http://www.ibm.com/xmlns/prod/websphere/business-process/callback-services/7.0/Binding
Interface de service Web de rappel de Business Flow Manager	BFMJAXWSCallbackInterface.wsdl	http://www.ibm.com/xmlns/prod/websphere/business-process/callback-services/7.0
Service Web de Human Task Manager	HTMJAXWSService.wsdl	http://www.ibm.com/xmlns/prod/websphere/human-task/services/7.0/Binding
Interface de service Web de Human Task Manager	HTMJAXWSInterface.wsdl	http://www.ibm.com/xmlns/prod/websphere/human-task/services/7.0
Types de données de service Web de Human Task Manager	HTMDataTypes.xsd	http://www.ibm.com/xmlns/prod/websphere/human-task/types/7.0
Service Web de rappel de Human Task Manager	HTMJAXWSCallbackService.wsdl	http://www.ibm.com/xmlns/prod/websphere/human-task/callback-services/7.0/Binding
Interface de service Web de rappel de Human Task Manager	HTMJAXWSCallbackInterface.wsdl	http://www.ibm.com/xmlns/prod/websphere/human-task/callback-services/7.0
Types de données communs de Business Process Choreographer	BPCDataTypes.xsd	http://www.ibm.com/xmlns/prod/websphere/bpc-common/types/7.0

API des services Web de Business Process Choreographer : valeurs standard

Utilisez les liens suivants pour retrouver les informations supplémentaires relatives aux valeurs standard applicables aux applications Web. Ces informations se trouvent sur des sites Internet non IBM dont le contenu technique est contrôlé par les détenteurs du site.

Ces liens sont fournis pour votre confort. Souvent, ils fournissent des informations qui ne sont pas spécifiques d'IBM WebSphere Process Server, mais utiles à la compréhension globale des services Web.

- Java API for XML-based Web Services (JAX-WS 2.0) (JSR-224; Java Community Process)
- Java Architecture for XML Binding (JAXB) 2.0 (JSR-222; Java Community Process)
- Web Services Description Language (WSDL) 1.1 (W3C)
- XML Schema Part 0: Primer Second Edition (W3C)
- XML Schema Part 1: Structures Second Edition (W3C)
- XML Schema Part 2: Datatypes Second Edition (W3C)
- Simple Object Access Protocol (SOAP) 1.1 (W3C)
- Web Services Policy Framework (WS-Policy) 1.5 (W3C)
- WS-Security 1.1 (OASIS)
- WS-Security UserName Token Profile 1.1 (OASIS)
- WS-AtomicTransaction 1.2 (OASIS)
- WS-Interoperability Basic Profile 1.1 (WS-Interoperability Organization)

Publication et exportation d'artefacts depuis l'environnement de serveur pour les applications client de services Web

Pour pouvoir développer des applications client en vue d'accéder aux API de services Web Business Process Choreographer, vous devez publier et exporter un certain nombre d'artefacts à partir de l'environnement de serveur WebSphere.

task_context

Les artefacts à exporter sont les suivants :

- Fichiers WSDL (Web Service Definition Language) décrivant le noeud final de service Web, les types de port et les opérations qui génèrent les API de services Web de Business Process Choreographer (toujours requis pour la génération de proxy de service Web).
- Fichiers XSD (XML Schema Definition) contenant des définitions de types de données référencés par des services dans les fichiers WSDL de Business Process Choreographer (toujours requis pour la génération de proxy de service Web).
- Fichiers WSDL et XSD décrivant les interfaces et les types de données de vos processus métier et tâches manuelles qui s'exécutent sur le serveur WebSphere. Ces fichiers supplémentaires sont requis uniquement si votre application client doit interagir directement avec les tâches manuelles ou les processus métier via les API de services Web. Ils ne sont pas nécessaires si votre application client est uniquement destinée à appeler des opérations qui peuvent être exécutées par Business Process Choreographer sans interaction directe avec vos instances de processus et de tâche, telles que l'émission de requêtes.
- Fichiers Web Service Policy (WS-Policy) décrivant la qualité des attributs de service pour l'API de services Web. Ils peuvent être exportés afin d'agir en tant que base pour la création des règles de service Web côté client.

  WS-Security
  Le message de demande doit contenir un jeton UserName ou un jeton LPTA.

  WS-Transaction
  Le message de demande peut contenir un contexte WS-

AtomicTransaction. Si le contexte est présent, la demande est traitée dans la portée des transactions de l'appelant.

Une fois ces artefacts publiés, vous devez les copier dans votre environnement de programmation client, dans lequel ils sont utilisés pour générer un proxy de service Web et des classes helper.

Tâches associées

«Publication des fichiers WSDL de Business Process Choreographer»
Un fichier WSDL (Web Service Definition Language) contient la description détaillée de toutes les opérations accessibles avec une API de services Web. Des fichiers WSDL séparés sont disponibles pour les API de services Web Business Flow Manager et Human Task Manager. Ces fichiers sont utilisés pour générer un proxy de service Web pour votre application.

«Exportation des fichiers WSDL et XSD pour les applications de services Web de processus métier et tâches manuelles», à la page 514
Les processus métier et les tâches manuelles disposent d'interfaces bien définies les rendant accessibles depuis l'extérieur en tant que services Web. Vous devez exporter les définitions d'interface WSDL et les définitions de type de données du schéma XML dans votre environnement de programmation client.

Publication des fichiers WSDL de Business Process Choreographer

Un fichier WSDL (Web Service Definition Language) contient la description détaillée de toutes les opérations accessibles avec une API de services Web. Des fichiers WSDL séparés sont disponibles pour les API de services Web Business Flow Manager et Human Task Manager. Ces fichiers sont utilisés pour générer un proxy de service Web pour votre application.

task_prereq

Avant de publier les fichiers, assurez-vous que l'adresse du noeud final de services Web correcte est spécifiée. Il s'agit de l'adresse URL qu'une application client utilise pour accéder aux API de services Web.

task_context

Vous devez publier ces fichiers WSDL ainsi que tous les fichiers XSD référencés par les fichiers WSDL. Vous pouvez ensuite les copier à partir de l'environnement WebSphere vers votre environnement de développement, où ils sont utilisés pour générer des classes helper et proxy de service Web. La publication des fichiers WSDL de Business Process Choreographer n'est nécessaire qu'une fois.

Tâches associées

«Publication du fichier WSDL des processus métier pour les applications de services Web»
La console d'administration permet de publier le fichier WSDL.

«Publication du fichier WSDL des tâches manuelles pour les applications de services Web», à la page 513
La console d'administration permet de publier le fichier WSDL.

Publication du fichier WSDL des processus métier pour les applications de services Web :

La console d'administration permet de publier le fichier WSDL.

task_procedure

1. Connectez-vous à la console d'administration avec un ID utilisateur titulaire des droits d'administrateur.
2. Cliquez sur **Applications → Modules SCA**.

   Remarque : Vous pouvez également cliquer sur **Applications → Types d'application → Applications d'entreprise WebSphere** pour visualiser une liste de toutes les applications d'entreprise disponibles.
3. Choisissez l'application **BPEContainer** dans la liste des applications ou modules SCA.
4. Sélectionnez l'option **Publier des fichiers WSDL** dans la liste des **Propriétés supplémentaires**
5. Cliquez sur le fichier .zip dans la liste.
6. Dans la fenêtre de téléchargement de fichiers qui s'affiche, cliquez sur **Enregistrer**.
7. Accédez à un dossier local et cliquez sur **Enregistrer**.

task_results

Le fichier .zip exporté est nommé `BPEContainer_nomnoeud_nomserveur_WSDLFiles.zip`. Il contient un fichier WSDL qui décrit les services Web, ainsi que tous les fichiers XSD référencés par le fichier WSDL.

Remarque : Le fichier .zip exporté contient les artefacts WSDL et XSD du service Web JAX-WS présenté dans la version 7 et du service Web JAX-RPC utilisé dans la version 6. Lorsque vous générez le proxy de service Web à l'aide de l'outil wsimport, vous sélectionnez les artefacts du service Web JAX-WS et les artefacts JAX-RPC sont ignorés.

Publication du fichier WSDL des tâches manuelles pour les applications de services Web :

La console d'administration permet de publier le fichier WSDL.

task_procedure

1. Connectez-vous à la console d'administration avec un ID utilisateur titulaire des droits d'administrateur.
2. Cliquez sur **Applications → Modules SCA**.

   Remarque : Vous pouvez également cliquer sur **Applications → Types d'application → Applications d'entreprise WebSphere** pour visualiser une liste de toutes les applications d'entreprise disponibles.
3. Choisissez l'application **TaskContainer** dans la liste des applications ou modules SCA.
4. Sélectionnez l'option **Publier des fichiers WSDL** dans la liste des **Propriétés supplémentaires**
5. Cliquez sur le fichier .zip dans la liste.
6. Dans la fenêtre de téléchargement de fichiers qui s'affiche, cliquez sur **Enregistrer**.
7. Accédez à un dossier local et cliquez sur **Enregistrer**.

task_results

Le fichier .zip exporté est nommé `TaskContainer_nomnoeud_nomserveur_WSDLFiles.zip`. Il contient un fichier WSDL qui décrit les services Web, ainsi que tous les fichiers XSD référencés par le fichier WSDL.

Remarque : Le fichier .zip exporté contient les artefacts WSDL et XSD du service Web JAX-WS présenté dans la version 7 et du service Web JAX-RPC utilisé dans la version 6. Lorsque vous générez le proxy de service Web à l'aide de l'outil wsimport, vous sélectionnez les artefacts du service Web JAX-WS et les artefacts JAX-RPC sont ignorés.

Exportation des fichiers WSDL et XSD pour les applications de services Web de processus métier et tâches manuelles

Les processus métier et les tâches manuelles disposent d'interfaces bien définies les rendant accessibles depuis l'extérieur en tant que services Web. Vous devez exporter les définitions d'interface WSDL et les définitions de type de données du schéma XML dans votre environnement de programmation client.

task_context

Cette procédure doit être répétée pour chaque processus métier ou tâche manuelle avec lequel/laquelle votre application client entre en interaction.

Par exemple, pour créer et démarrer une tâche manuelle, les éléments d'information suivants doivent être transmis à l'instance de tâche :

- Le nom du modèle de tâche
- L'espace de nom du modèle de tâche.
- Un message d'entrée contenant les données métier mises en forme
- Un encapsuleur de réponse pour le renvoi du message de réponse
- Un message d'erreur pour le renvoi des erreurs et des exceptions

Ces éléments sont encapsulés dans un objet métier unique. Toutes les opérations de l'interface du service Web sont modélisées sous forme d'opération document/littéral encapsulé. Les paramètres d'entrée et de sortie relatifs à ces opérations sont encapsulés dans des documents d'encapsulation. Les autres objets métier définissent la réponse correspondante et les formats des messages d'erreur.

Pour permettre la création et le démarrage du processus métier ou de la tâche manuelle via un service Web, l'application client côté client doit pouvoir accéder à ces objets d'encapsulation.

Pour cela, il suffit d'exporter les objets métier depuis l'environnement WebSphere sous forme de fichiers WSDL (Web Service Definition Language) et XSD (XML Schema Definition), en important les définitions des types de données dans votre environnement de programmation.

task_procedure

1. Lancez l'espace de travail WebSphere Integration Developer s'il n'est pas déjà en cours d'exécution.
2. Sélectionnez le module de bibliothèque contenant les objets métier à exporter. Un module de bibliothèque est un fichier compressé contenant les objets métier requis.

3. Exportez le module de bibliothèque.
4. Copiez les fichiers exportés vers votre environnement de développement d'applications client.

Exemple

En supposant qu'un processus métier expose l'opération de service Web suivante :

```
<wsdl:operation name="updateCustomer">
    <wsdl:input message="tns:updateCustomerRequestMsg" name="updateCustomerRequest"/>
    <wsdl:output message="tns:updateCustomerResponseMsg" name="updateCustomerResponse"/>
    <wsdl:fault message="tns:updateCustomerFaultMsg" name="updateCustomerFault"/>
  </wsdl:operation>
```

avec les messages WSDL définis comme suit :

```
<wsdl:message name="updateCustomerRequestMsg">
    <wsdl:part element="types:updateCustomer" name="updateCustomerParameters"/>
  </wsdl:message>
  <wsdl:message name="updateCustomerResponseMsg">
    <wsdl:part element="types:updateCustomerResponse" name="updateCustomerResult"/>
  </wsdl:message>
  <wsdl:message name="updateCustomerFaultMsg">
    <wsdl:part element="types:updateCustomerFault" name="updateCustomerFault"/>
  </wsdl:message>
```

Les éléments *concrets* définis par l'utilisateur `types:updateCustomer`, `types:updateCustomerResponse` et `types:updateCustomerFault` doivent être transmis vers et depuis les API de services Web au moyen des paramètres `UserData` dans toutes les opérations *génériques* (call, sendMessage, etc.) exécutées par l'application client.

Ces éléments définis par le client sont créés, sérialisés et désérialisés côté application client à l'aide des classes générées par les fichiers XSD exportés . La génération de ces classes fait partie de la génération du proxy de service Web dans lequel les fichiers WSDL et XSD exportés sont inclus.

Les opérations génériques de l'interface des services Web propagent l'élément encapsuleur du document vers et depuis l'opération implémentée par le processus métier ou la tâche utilisateur. Pour l'exemple d'opération de l'exemple précédent, un message SOAP de service Web peut se présenter comme suit :

```
<soapenv:Envelope xmlns:soapenv="..." ...>
  <soapenv:Header>
    ...
  </soapenv:Header>
  <soapenv:Body>
    <bfm:sendMessage
         xmlns:bfm="http://www.ibm.com/xmlns/prod/websphere/business-process/services/7.0">
      <processTemplateName>customerProcessTemplate</processTemplateName>
      <portType xmlns:cns="http://example.com/customerProcess">cns:customerProcessPortType</portType>
      <operation>updateCustomer</operation>
      <input>
        <cns:updateCustomer xmlns:cns="http://example.com/customerProcess">
          <street>1600 Pennsylvania Avenue Northwest</street>
          <city>Washington, DC 20006</city>
        </cns:updateCustomer>
      </input>
    </bfm:sendMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

Développement d'applications client dans l'environnement de services Web Java

Vous pouvez utiliser tout environnement de développement Java compatible avec les services Web Java en vue de développer des applications client pour les API de services Web Business Process Choreographer.

Tâches associées

«Génération d'un proxy de services Web (services Web Java)»
Les applications client de services Web Java utilisent un *proxy de service Web* pour interagir avec les API de services Web Business Process Choreographer.

«Création d'une application client pour les processus métier et les tâches manuelles (services Web Java)», à la page 519
Une application client envoie des requêtes aux API de services Web Business Process Choreographer et reçoit des réponses de ces dernières. En utilisant un client proxy de service Web pour gérer les communications et des classes helper pour formater les types de données complexes, une application client peut appeler les méthodes de service Web comme s'il s'agissait de fonctions locales.

Génération d'un proxy de services Web (services Web Java)

Les applications client de services Web Java utilisent un *proxy de service Web* pour interagir avec les API de services Web Business Process Choreographer.

task_context

Un proxy de service Web destiné aux services Web Java contient plusieurs classes JavaBeans qui sont appelées par l'application client pour exécuter des demandes de services Web. Le proxy de service Web gère l'assemblage des paramètres de services sous forme de messages SOAP, envoie les messages SOAP au service Web via HTTP, reçoit les réponses du service Web et transmet toutes les données renvoyées à l'application client.

Par conséquent, un proxy de service Web permet à une application client d'appeler un service Web comme s'il s'agissait d'une fonction locale.

Remarque : La génération d'un proxy de service Web n'est nécessaire qu'une fois. Toutes les applications client accédant aux mêmes API de services Web peuvent alors utiliser le même proxy de service Web.

Dans l'environnement de services Web IBM, vous pouvez générer un proxy de service Web de l'une des manières suivantes.

- A l'aide des environnements de développement intégrés Rational Application Developer ou WebSphere Integration Developer.
- A l'aide de l'outil de ligne de commande wsimport.

Les autres environnements de développement de services Web Java comprennent généralement l'outil wsimport ou des fonctions de génération d'applications client propriétaires.

Tâches associées

«Utilisation de Rational Application Developer afin de générer un proxy de service Web pour une application de services Web»
Vous pouvez utiliser l'environnement de développement intégré Rational Application Developer afin de générer un proxy de service Web pour votre application client de services Web. La séquence d'étapes suivante s'applique à Rational Application Developer version 7.5.3.

«Utilisation de l'outil de ligne de commande wsimport afin de générer un proxy de service Web pour une application de services Web», à la page 518
Vous pouvez utiliser l'outil de ligne de commande wsimport afin de générer un proxy de service Web pour une application de services Web.

Utilisation de Rational Application Developer afin de générer un proxy de service Web pour une application de services Web :

Vous pouvez utiliser l'environnement de développement intégré Rational Application Developer afin de générer un proxy de service Web pour votre application client de services Web. La séquence d'étapes suivante s'applique à Rational Application Developer version 7.5.3.

task_prereq

Avant de générer un proxy de service Web, vous devez avoir préalablement exporté les fichiers WSDL et XSD décrivant les interfaces de services Web des tâches manuelles ou des processus métier depuis l'environnement WebSphere, puis les avoir copiés dans votre environnement de programmation client.

task_procedure

1. Ajoutez à votre projet le fichier WSDL approprié.
   - Pour les processus métier :
     a. Décompressez le fichier exporté `BPEContainer_nomnoeud_nomserveur_WSDLFiles.zip` dans un répertoire temporaire. Ne modifiez pas le contenu de ce répertoire et gardez à l'esprit que seuls les fichiers WSDL et XSD suivants sont utilisés pour générer le proxy de service Web pour les interactions avec les processus métier :
        – BFMJAXWSService.wsdl
        – BFMJAXWSInterface.wsdl
        – BFMJAXWSCallbackService.wsdl
        – BFMJAXWSCallbackInterface.wsdl
        – BFMDataTypes.xsd
        – BPCDataTypes.xsd
        – wsa.xsd
     b. Importez le sous-répertoire `META-INF` à partir du répertoire décompressé `BPEContainer_`*`nomnoeud_nomserveur`*`.ear/bfmjaxws.jar`.
   - Pour les tâches manuelles :
     a. Décompressez le fichier exporté `TaskContainer_nomnoeud_nomserveur_WSDLFiles.zip` dans un répertoire temporaire. Ne modifiez pas le contenu de ce répertoire et gardez à l'esprit que seuls les fichiers WSDL et XSD suivants sont utilisés pour générer le proxy de service Web pour les interactions avec les tâches manuelles :

- HTMJAXWSService.wsdl
- HTMJAXWSInterface.wsdl
- HTMJAXWSCallbackService.wsdl
- HTMJAXWSCallbackInterface.wsdl
- HTMDataTypes.xsd
- BPCDataTypes.xsd
- wsa.xsd

b. Importez le sous-répertoire `META-INF` à partir du répertoire décompressé `TaskContainer_`*`nomnoeud_nomserveur`*`.ear/htmjaxws.jar`.

Un nouveau répertoire `wsdl` et une structure de sous-répertoire sont créés dans votre projet.

2. Sélectionnez le fichier `BFMJAXWSService.wsdl` situé dans le répertoire `wsdl` nouvellement créé.
3. Cliquez avec le bouton droit et sélectionnez **Web Services (Services web)** → **Generate client (Générer un client)**.

   Avant d'entamer le reste de la procédure, assurez-vous que le serveur a démarré.
4. Dans la fenêtre Web services (Services web), cliquez sur **Next (Suivant)** afin d'accepter toutes les valeurs par défaut.
5. Dans la fenêtre de configuration du client de service Web de JAX-WS des services Web, modifiez la version du code JAX-WS pour une génération en 2.0 et cliquez sur **Terminer** pour accepter toutes les autres valeurs par défaut.
6. Répétez les étapes 2 à 5 de cette procédure avec `HTMJAXWSService.wsdl` et remplacez tous les fichiers si vous y êtes invité.

task_results

Un proxy de service Web, constitué de plusieurs classes proxy, locator et JAXB est généré et ajouté à votre projet.

Utilisation de l'outil de ligne de commande wsimport afin de générer un proxy de service Web pour une application de services Web :

Vous pouvez utiliser l'outil de ligne de commande wsimport afin de générer un proxy de service Web pour une application de services Web.

task_prereq

Avant de générer un proxy de service Web, vous devez avoir préalablement exporté les fichiers WSDL décrivant les API de services Web des tâches manuelles ou des processus métier depuis l'environnement WebSphere, puis les avoir copiés dans votre environnement de programmation client.

task_procedure

1. Générez un proxy de service Web pour les API de services Web de Business Process Choreographer :

   Remarque : Pour une description détaillées de l'outil de ligne de commande wsimport des applications JAX-WS, voir la documentation sur la ligne de commande wsimport de WebSphere Application Server.

```
wsimport.bat BFMJAXWSService.wsdl myService1.wsdl myService2.wsdl
 –d proxy-bfm
 -wsdllocation <bfm_location>

wsimport.bat HTMJAXWSService.wsdl myService1.wsdl myService2.wsdl
 –d proxy-htm
 -wsdllocation <htm_location>
```

Dans cet exemple, `myService1.wsdl` et `myService2.wsdl` contiennent les définitions d'interface des processus métier personnalisés, ou des tâches manuelles, ou des deux. De plus, *<bfm_location>* et *<htm_location>* peuvent être obtenus à partir de l'élément *<port>* de WSDL, respectivement dans `BFMJAXWSService.wsdl` et dans `HTMJAXWSService.wsdl`.

Vous pouvez fusionner les deux proxys dans un répertoire commun (par exemple, proxy-bpc) et remplacer les fichiers existants si vous y êtes invité.

2. Incluez à votre projet les fichiers classe générés.

Tâches associées

«Création d'une application client pour les processus métier et les tâches manuelles (services Web Java)»
Une application client envoie des requêtes aux API de services Web Business Process Choreographer et reçoit des réponses de ces dernières. En utilisant un client proxy de service Web pour gérer les communications et des classes helper pour formater les types de données complexes, une application client peut appeler les méthodes de service Web comme s'il s'agissait de fonctions locales.

Création d'une application client pour les processus métier et les tâches manuelles (services Web Java)

Une application client envoie des requêtes aux API de services Web Business Process Choreographer et reçoit des réponses de ces dernières. En utilisant un client proxy de service Web pour gérer les communications et des classes helper pour formater les types de données complexes, une application client peut appeler les méthodes de service Web comme s'il s'agissait de fonctions locales.

task_prereq

Avant de commencer à créer une application client, générez le proxy de service Web.

task_context

Vous pouvez développer des applications client à l'aide de tout outil de développement compatible avec les services Web, par exemple, IBM Rational Application Developer. Vous pouvez créer tous types d'applications de services Web pour appeler les API de services Web génériques.

task_procedure

1. Créez un projet d'application client.
2. Générez le proxy de service Web.
3. Codez votre application client.
4. Générez le projet.
5. Exécutez l'application client.

task_example

L'exemple suivant illustre comment utiliser l'API de services Web Business Flow Manager.

```
try {
   // create bfm proxy
   BFMJAXWSPortType bfm = new BFMJAXWSService().getBFMJAXWSPort();

   // call getProcessTemplate
   ProcessTemplateType ptt =
     bfm.getProcessTemplate("MY_PROCESS_TEMPLATE_NAME");

   // handle return value
   System.out.println("Process template '" + ptt.getName() + "' found, details following:");
   System.out.println("Execution mode: " +
       ptt.getExecutionMode());
   System.out.println("Schema version:" + ptt.getSchemaVersion());
 } catch (Exception e) {
   if ( e instanceof ProcessFaultMsg )
   {
     ProcessFaultMsg pfm = (ProcessFaultMsg) e;
     List<FaultStackType> list =
         ( pfm.getFaultInfo() ).getFaultStack();
     FaultStackType fault = list.get( 0 );
     System.out.println( "ProcessFaultMessage: " +
         fault.getMessage() );
   }
   else
   {
     e.printStackTrace( System.out );
   }
 }
```

Tâches associées

«Utilisation de l'outil de ligne de commande wsimport afin de générer un proxy de service Web pour une application de services Web», à la page 518
Vous pouvez utiliser l'outil de ligne de commande wsimport afin de générer un proxy de service Web pour une application de services Web.

«Génération d'un proxy de services Web (services Web Java)», à la page 516
Les applications client de services Web Java utilisent un *proxy de service Web* pour interagir avec les API de services Web Business Process Choreographer.

Ajout de sécurité

Le service Web de Business Process Choreographer nécessite que vous configuriez votre application client pour un mécanisme d'authentification.

task_context

Par défaut, Business Process Choreographer prend en charge les mécanismes d'authentification suivants :

Jeton Username (nom d'utilisateur)
Un client de service Web fournit un jeton Username comme moyen d'identifier le demandeur par son "nom d'utilisateur", et en utilisant en option un mot de passe permettant d'authentifier cette identité sur le fournisseur de services Web.

Jeton de sécurité binaire – Jeton LTPA (Lightweight Third-Party Authentication)
Un client de service Web fournit un jeton LTPA comme moyen d'authentifier le demandeur sur le fournisseur de services Web.

Vous pouvez remplacer les règles de sécurité du service Web de Business Process Choreographer par un autre mécanisme d'authentification. Cependant, il est impossible d'appeler les opérations du service Web de Business Process

Choreographer comme un utilisateur non authentifié. Ainsi, un mécanisme d'authentification est toujours requis.

Ajout d'un support de transaction

Les applications client de service Web peuvent être configurées pour permettre au traitement de la requête côté serveur de participer à la transaction client, en transmettant un contexte d'application client en tant que requête de service. Ce support de transaction atomique est défini dans la spécification Web Services-Atomic Transaction (WS-AT).

task_context

Business Process Choreographer exécute chaque demande d'opération de service Web comme une transaction globale distincte. Les applications client peuvent être configurées en vue d'utiliser un support de transaction pour :

- Propagez le contexte de transaction du client. Le traitement des requêtes côté serveur est effectué dans le contexte de transaction de l'application client et est donc validé (ou annulé) avec la transaction du client. A l'inverse, si le serveur rencontre un problème alors que le service Web est en cours d'exécution et demande une invalidation, la transaction de l'application client est également invalidée.
- Ne pas utiliser de prise en charge de la transaction. Business Process Choreographer crée une nouvelle transaction globale afin d'exécuter la requête, mais le traitement de la requête côté serveur n'est pas effectué au moyen du contexte de transaction de l'application client.

Les règles de service Web associées au service Web de Business Process Choreographer permet à chaque message de requête de contenir un contexte de transaction WS-AT, tel que décrit ci-dessus. Si vous choisissez d'appeler les opérations du service Web sans passer un contexte de transaction client, il est conseillé d'ignorer les règles de la transaction côté fournisseur et de configurer le client du service Web sans aucune règle de transaction.

Développement d'applications client à l'aide de l'API JMS de Business Process Choreographer

Vous pouvez développer des applications client accédant aux applications de processus métier de façon asynchrone grâce à l'API JMS (Java Messaging Service).

task_context

Les applications client JMS échangent des messages de demande et de réponse avec l'API JMS. Pour créer un message de demande, l'application client remplit le corps du message JMS TextMessage avec un élément XML représentant l'encapsuleur document/littéral de l'opération correspondante.

Concepts associés

«Exigences des processus métier»
Les processus métier développés au moyen de WebSphere Integration Developer pour être exécutés dans l'application Business Process Choreographer doivent être conformes à des règles spécifiques afin d'être accessibles via l'API JMS.

«Autorisation pour les affichages JMS»
Pour autoriser l'accès à l'interface JMS, des paramètres de sécurité doivent être activés dans WebSphere Application Server.

Tâches associées

«Accès à l'interface JMS», à la page 523
Pour envoyer et recevoir des messages par le biais de l'interface JMS, une application doit d'abord créer une connexion au bus BPC.cellname.Bus, créer une session, puis générer des expéditeurs et des destinataires de message.

«Copie d'artefacts pour les applications client JMS», à la page 527
Un certain nombre d'artefacts peuvent être copiés à partir de l'environnement WebSphere Process Server pour faciliter la création d'applications client JMS.

«Vérification du message de réponse pour les exceptions de métier», à la page 528
Les applications client JMS doivent vérifier l'en-tête de message de tous les messages de réponse pour les exceptions de métier.

«Exemple : exécution d'un processus de longue durée à l'aide de l'API JMS de Business Process Choreographer», à la page 528
Cet exemple montre comment créer une application client générique utilisant l'API JMS pour exploiter des processus de longue durée.

Exigences des processus métier

Les processus métier développés au moyen de WebSphere Integration Developer pour être exécutés dans l'application Business Process Choreographer doivent être conformes à des règles spécifiques afin d'être accessibles via l'API JMS.

Les exigences sont les suivantes :

1. Les interfaces des processus métier doivent être définies à l'aide du style "document/literal wrapped" défini dans l'API Java pour la spécification XML-RPC (JAX-RPC 1.1). Il s'agit du style par défaut défini pour l'ensemble des processus métier et des tâches manuelles développés avec WebSphere Integration Developer.
2. Les messages d'erreur accessibles aux processus métier et aux tâches manuelles des opérations de service Web doivent comprendre un seul composant de message WSDL défini au moyen d'un élément de schéma XML. Par exemple :

   ```
   <wsdl:part name="myFault" element="myNamespace:myFaultElement"/>
   ```

Information associée

Page de téléchargement d'API Java pour XML-RPC (JAX-RPC)

Quel style de langage WSDL dois-je utiliser ?

Autorisation pour les affichages JMS

Pour autoriser l'accès à l'interface JMS, des paramètres de sécurité doivent être activés dans WebSphere Application Server.

Lorsque le conteneur de processus métier est installé, le rôle **JMSAPIUser** doit être mappé avec un ID utilisateur. Cet ID utilisateur permet d'émettre toutes les

demandes de l'API JMS. Par exemple, si **JMSAPIUser** est mappé avec "Utilisateur A", toutes les demandes de l'API JMS apparaissent dans le moteur de processus avec pour origine "Utilisateur A".

Le rôle **JMSAPIUser** doit être affecté aux autorités suivantes :

Demande	Autorisation requise
forceTerminate	Administrateur de processus
sendEvent	Propriétaire potentiel d'activité ou administrateur de processus

Remarque : Pour toutes les demandes, aucune autorisation spéciale n'est requise.

L'autorité spéciale est accordée à une personne avec le rôle d'administrateur de processus métier. Un administrateur de processus métier est un rôle spécial. Il est différent de celui de l'administrateur de processus d'une instance de processus. Il dispose de tous les privilèges.

Vous ne pouvez pas supprimer l'ID utilisateur du lanceur de processus à partir de votre registre des utilisateurs alors que l'instance du processus existe. Si vous supprimez cet ID utilisateur, la navigation dans ce processus ne peut se poursuivre. Vous recevrez l'exception suivante dans le fichier journal du système :

```
no unique ID for: <ID utilisateur>
```

Accès à l'interface JMS

Pour envoyer et recevoir des messages par le biais de l'interface JMS, une application doit d'abord créer une connexion au bus BPC.cellname.Bus, créer une session, puis générer des expéditeurs et des destinataires de message.

task_context

Le serveur de processus accepte les messages Java Message Service (JMS) qui suivent le paradigme point-à-point. Une application qui envoie ou qui reçoit des messages JMS doit exécuter les actions suivantes.

L'exemple suivant suppose que le client JMS est exécuté dans un environnement géré (Enterprise JavaBeans, client d'application ou conteneur de client Web).

task_procedure

1. Créez une connexion au BPC.*nomcellule*.Bus. Il n'existe pas de fabrique de connexions préconfigurée pour les requêtes d'une application client : l'application client peut soit utiliser la commande ReplyConnectionFactory de l'API JMS, soit créer sa propre fabrique de connexions, auquel cas elle peut utiliser la recherche JNDI (Java Naming and Directory Interface) pour récupérer la fabrique de connexions. Le nom de recherche JNDI doit être identique au nom indiqué lors de la configuration de la file d'attente des demandes externes de Business Process Choreographer. L'exemple suivant suppose que l'application client crée sa propre fabrique de connexions nommée "jms/clientCF".

   ```
   //Obtain the default initial JNDI context.
   Context initialContext = new InitialContext();

   // Look up the connection factory.
   // Create a connection factory that connects to the BPC bus.
   ```

```
// Call it, for example, "jms/clientCF".
// Also configure an appropriate authentication alias.
ConnectionFactory connectionFactory =
            (ConnectionFactory)initialcontext.lookup("jms/clientCF");

// Create the connection.
Connection connection = connectionFactory.createConnection();
```

2. Créez une session afin de pouvoir créer les expéditeurs et les destinataires de message.

```
// Create a transaction session using auto-acknowledgment.
Session session = connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
```

3. Créez un expéditeur de message pour envoyer les messages. Le nom de recherche JNDI doit être identique au nom indiqué lors de la configuration de la file d'attente des demandes externes de Business Process Choreographer.

```
// Look up the destination of the Business Process Choreographer input queue to
// send messages to.
Queue sendQueue = (Queue) initialcontext.lookup("jms/BFMJMSAPIQueue");

// Create a message producer.
MessageProducer producer = session.createProducer(sendQueue);
```

4. Créez un destinataire de message pour recevoir les réponses. Le nom de recherche JNDI de la destination de la réponse peut indiquer une destination définie par l'utilisateur, mais il peut également indiquer la destination de la réponse par défaut (définie par Business Process Choreographer) jms/BFMJMSReplyQueue. Dans les deux cas, la destination de la réponse doit être basée sur BPC.<cellname>.Bus.

```
// Look up the destination of the reply queue.
Queue  replyQueue = (Queue) initialcontext.lookup("jms/BFMJMSReplyQueue");

// Create a message consumer.
MessageConsumer consumer = session.createConsumer(replyQueue);
```

5. Envoyez un message.

```
// Start the connection.
connection.start();

// Create a message - see the task descriptions for examples - and send it.
// This method is defined elsewhere ...
String payload = createXMLDocumentForRequest();
TextMessage requestMessage = session.createTextMessage(payload);

// Set mandatory JMS header.
// targetFunctionName is the operation name of JMS API
// (for example, getProcessTemplate, sendMessage)
requestMessage.setStringProperty("TargetFunctionName", targetFunctionName);

// Set the reply queue; this is mandatory if the replyQueue
// is not the default queue (as it is in this example).
requestMessage.setJMSReplyTo(replyQueue);

// Send the message.
producer.send(requestMessage);

// Get the message ID.
String jmsMessageID = requestMessage.getJMSMessageID();

session.commit();
```

6. Recevez la réponse.

```
// Receive the reply message and analyse the reply.
TextMessage  replyMessage = (TextMessage) consumer.receive();
```

```
// Get the payload.
String payload = replyMessage.getText();

session.commit();
```

7. Mettez fin à la connexion, puis libérez les ressources.

```
 // Final housekeeping; free the resources.
session.close();
connection.close();
```

Remarque : Vous n'êtes pas obligé de mettre fin à la connexion après chaque transaction. Une fois la connexion démarrée, vous pouvez échanger n'importe quel nombre de messages de demande et de réponse avant de mettre fin à la connexion. L'exemple illustre un cas simple avec un appel unique au sein d'une méthode métier unique.

Concepts associés

«Structure d'un message JMS de Business Process Choreographer»
L'en-tête et le corps d'un message JMS doivent avoir une structure prédéfinie.

Structure d'un message JMS de Business Process Choreographer

L'en-tête et le corps d'un message JMS doivent avoir une structure prédéfinie.

Un message JMS (Java Message Service) se compose des éléments suivants :

- Un en-tête de message pour l'identification du message et l'acheminement de l'information.
- Le corps (charge) du message qui renferme le contenu.

Business Process Choreographer ne prend en charge que les formats de message texte.

En-tête de message

JMS permet aux clients d'accéder à certains champs d'en-tête de message.

Les champs d'en-tête suivants peuvent être définis par un client JMS de Business Process Choreographer :

JMSReplyTo
Destination à laquelle est envoyée la réponse à une requête. Si ce champ n'est pas spécifié dans le message de requête, la réponse est alors envoyée à la destination de réponse par défaut de l'interface d'exportation (l'exportation correspond à l'affichage de l'interface client d'un composant de processus métier). Il est possible d'obtenir cette destination à l'aide de `initialContext.lookup("jms/BFMJMSReplyQueue");`

TargetFunctionName
Le nom de l'opération WSDL pourrait être "queryProcessTemplates", par exemple. Ce champ doit toujours être défini. Notez que TargetFunctionName spécifie l'opération de l'interface du message JMS générique décrite ici. A ne pas confondre avec les opérations fournies par des tâches ou des processus concrets pouvant être appelés indirectement à l'aide de l'opération call ou sendMessage, par exemple.

Un client Business Process Choreographer peut également accéder aux champs d'en-tête suivants :

JMSMessageID
Identifie un message de manière unique. Défini par le fournisseur JMS lorsque le message est envoyé. Si le client définit le champ JMSMessageID avant l'envoi du message, il est systématiquement remplacé par le fournisseur JMS. Si l'ID du message est requis à des fins d'authentification, le client peut alors obtenir le paramètre JMSMessageID après l'envoi du message.

JMSCorrelationID
Relie les messages. Ne pas définir ce champ. Un message de réponse Business Process Choreographer contient toujours le champ JMSMessageID du message de requête.

Chaque message de réponse contient les champs d'en-tête JMS suivants :

- **IsBusinessException**

  "False" pour les messages de sortie WSDL ou "True" pour les messages d'erreur WSDL.

Les exceptions ServiceRuntimeExceptions ne sont pas renvoyées aux applications client asynchrones. Lorsqu'une exception sévère se produit lors du traitement d'un message de requête JMS, une erreur d'exécution est générée, ce qui provoque l'annulation de la transaction en cours de traitement. Le message de requête JMS est alors redistribué. Si l'erreur se produit prématurément dans la phase d'exportation SCA du traitement du message (par exemple, lors de sa désérialisation), de nouvelles tentatives sont exécutées jusqu'au nombre maximum de livraisons échouées spécifié par la destination de réception de la fonction d'exportation SCA. Une fois ce nombre atteint, le message de requête est ajouté à la destination d'exception système du bus Business Process Choreographer. Cependant, si l'échec se produit lors du traitement réel de la requête par le composant SCA de Business Flow Manager, le message de requête échoué est géré par l'infrastructure de gestion des événements en échec de WebSphere Process Server, autrement dit, on se retrouve dans la base de données de gestion des événements échoués si les tentatives ne permettent pas de résoudre la situation exceptionnelle.

Corps du message

Les opérations exposées par les processus business ou les tâches utilisateur doivent respecter le style d'encapsuleur du document/littéral. Le corps du message JMS est une chaîne contenant un document XML qui représente l'élément encapsuleur du document/littéral de l'opération. Les opérations génériques de l'interface des messages JMS propagent l'élément encapsuleur du document vers et depuis l'opération implémentée par le processus métier ou la tâche utilisateur.

L'exemple suivant illustre un corps de message de demande valide simple :

```
<bfm:queryProcessTemplates
     xmlns:bfm="http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0">
  <whereClause>PROCESS_TEMPLATE.STATE IN (1)</whereClause>
</bfm:queryProcessTemplates>
```

L'exemple suivant illustre un corps de message de demande valide plus complexe. L'application client dispose d'une opération d'API sendMessage pour soumettre un message à un processus spécifique. Le message d'entrée du processus correspond à l'un des paramètres de l'API ; ce message représente le message d'entrée d'une opération métier exposée par un processus client. Le processus contient une activité de réception qui consomme le message.

L'élément `bfm:sendMessage` correspond à l'élément encapsuleur du document de l'opération d'API JMS. Il inclut l'élément `cns:updateCustomer`, qui correspond à l'élément encapsuleur du document de l'opération implémentée par le processus. Ce processus possède par exemple une activité `bpel:receive` qui fait référence au type de port WSDL `cns:customerProcessPortType` et l'opération WSDL `updateCustomer`.

```
<bfm:sendMessage
     xmlns:bfm="http://www.ibm.com/xmlns/prod/websphere/business-process/services/6.0">
  <processTemplateName>customerProcessTemplate</processTemplateName>
  <portType xmlns:cns="http://example.com/customerProcess">cns:customerProcessPortType</portType>
  <operation>updateCustomer</operation>
  <cns:updateCustomer xmlns:cns="http://example.com/customerProcess">
    <street>1600 Pennsylvania Avenue Northwest</street>
    <city>Washington, DC 20006</city>
  </cns:updateCustomer>
</bfm:sendMessage>
```

Tâches associées

«Vérification du message de réponse pour les exceptions de métier», à la page 528
Les applications client JMS doivent vérifier l'en-tête de message de tous les messages de réponse pour les exceptions de métier.

Copie d'artefacts pour les applications client JMS

Un certain nombre d'artefacts peuvent être copiés à partir de l'environnement WebSphere Process Server pour faciliter la création d'applications client JMS.

task_context

Ces artefacts sont obligatoires uniquement si vous utilisez BOXMLSerializer pour créer le corps du message JMS. Pour l'API JMS, ces artefacts sont :

`BFMIF.wsdl`

`BFMIF.xsd`

`BPCGen.xsd`

`wsa.xsd`

Vous devez publier et exporter ces fichiers à par tir de l'environnement WebSphere Process Server vers votre environnement de développement.

Tâches associées

«Publication du fichier WSDL des processus métier pour les applications JMS»
La console d'administration permet de publier le fichier WSDL.

Publication du fichier WSDL des processus métier pour les applications JMS

La console d'administration permet de publier le fichier WSDL.

task_procedure

1. Connectez-vous à la console d'administration avec un ID utilisateur titulaire des droits d'administrateur.
2. Cliquez sur **Applications → Modules SCA**.

   Remarque : Vous pouvez également cliquer sur **Applications → Types d'application → Applications d'entreprise WebSphere** pour visualiser une liste de toutes les applications d'entreprise disponibles.
3. Choisissez l'application **BPEContainer** dans la liste des applications ou modules SCA.

4. Sélectionnez l'option **Publier des fichiers WSDL** dans la liste des **Propriétés supplémentaires**
5. Cliquez sur le fichier .zip dans la liste.
6. Dans la fenêtre de téléchargement de fichiers qui s'affiche, cliquez sur **Enregistrer**.
7. Accédez à un dossier local et cliquez sur **Enregistrer**.

task_results

Le fichier .zip exporté s'appelle `BPEContainer_WSDLFiles.zip`. Il contient un fichier WSDL ainsi que tous les fichiers XSD référencés par le fichier WSDL.

Vérification du message de réponse pour les exceptions de métier

Les applications client JMS doivent vérifier l'en-tête de message de tous les messages de réponse pour les exceptions de métier.

task_context

Une application client JMS doit d'abord vérifier la propriété **IsBusinessException** de l'en-tête du message de réponse.

Par exemple :

task_example

```
// receive response message
Message receivedMessage = ((JmsProxy) getToBeInvokedUponObject().receiveMessage();
String strResponse = ((TextMessage) receivedMessage).getText();

if (receivedMessage.getStringProperty("IsBusinessException") {
  // strResponse is a bussiness fault
  // any api can end w/a processFaultMsg
  // the call api also w/a businessFaultMsg
 }
else {
  // strResponse is the output message
}
```

Concepts associés

«Structure d'un message JMS de Business Process Choreographer», à la page 525
L'en-tête et le corps d'un message JMS doivent avoir une structure prédéfinie.

Exemple : exécution d'un processus de longue durée à l'aide de l'API JMS de Business Process Choreographer

Cet exemple montre comment créer une application client générique utilisant l'API JMS pour exploiter des processus de longue durée.

task_procedure

1. Configurez l'environnement JMS, comme décrit dans «Accès à l'interface JMS», à la page 523.
2. Obtenez une liste des définitions de processus installées.
   - Envoyez la commande `queryProcessTemplates`.
   - Cette commande renvoie une liste d'objets ProcessTemplate.

3. Obtenez une liste d'activités de démarrage (activités de réception ou de sélection avec `createInstance="yes"`).
   - Envoyez `getStartActivities`.
   - Cette commande renvoie une liste d'objets InboundOperationTemplate.
4. Créez un message d'entrée. Ce message est propre à l'environnement et peut nécessiter l'emploi d'artefacts prédéployés, propres à chaque processus.
5. Créez une instance de processus.
   - Emettez une instruction `sendMessage`.

   Grâce à l'API JMS, vous pouvez également utiliser l'opération `call` pour l'interaction avec des opérations de demande-réponse de longue durée fournies par un processus métier. Cette opération renvoie le résultat ou l'erreur d'opération à la destination de réponse spécifiée, même après une longue période. Par conséquent, si vous utilisez l'opération `call`, il n'est pas nécessaire d'utiliser les opérations `query` et `getOutputMessage` pour que le message de sortie de processus ou d'erreur s'affiche.
6. OptionalColonSymbol Obtenez les messages de sortie des instances de processus en répétant la procédure suivante :
   a. Emettez la commande `query` pour obtenir l'état achevé de l'instance de processus.
   b. Emettez la commande `getOutputMessage` .
7. OptionalColonSymbol Travaillez maintenant sur les opérations supplémentaires présentées par le processus :
   a. Envoyez les commandes `getWaitingActivities` ou `getActiveEventHandlers` pour obtenir une liste des objets InboundOperationTemplate.
   b. Créez des messages d'entrée.
   c. Envoyez les messages à l'aide de la commande `sendMessage`.
8. OptionalColonSymbol Obtenez et définissez des propriétés personnalisées définies sur le processus ou les activités qu'il contient, en utilisant les commandes `getCustomProperties` et `setCustomProperties`.
9. Terminez le travail sur l'instance de processus :
   a. Envoyez `delete` et `terminate` pour mettre fin au processus de longue durée.

Développement d'applications Web pour les processus métier et tâches manuelles à l'aide de composants JSF

Business Process Choreographer offre un certain nombre de composants JavaServer Faces (JSF). Vous pouvez étendre et intégrer ces composants pour ajouter une fonction de processus métier et de tâches manuelles à des applications Web.

task_context

WebSphere Integration Developer permet de générer une application Web. Pour les applications comprenant des tâches manuelles, vous pouvez générer un client JSF personnalisé. Pour plus d'informations sur la génération d'un client JSF, consultez le centre de documentation de WebSphere Integration Developer.

Vous pouvez également développer votre client Web à l'aide des composants JSF fournis par Business Process Choreographer.

task_procedure

1. Créez un projet dynamique et modifiez les propriétés Web Project Features pour inclure les composants de base JSF.

   Pour plus d'informations sur la création d'un projet Web, accédez au centre de documentation de WebSphere Integration Developer.

2. Ajoutez les fichiers archive Java (JAR) préalables de Business Process Choreographer Explorer.

   Ajoutez les fichiers suivants au répertoire `WEB-INF/lib` de votre projet :

   - bpcclientcore.jar
   - bfmclientmodel.jar
   - htmclientmodel.jar
   - bpcjsfcomponents.jar

   Dans WebSphere Process Server, ces fichiers se trouvent tous dans le répertoire suivant :

   - Windows Sous les plateformes Windows® : *racine_installation*`\ProcessChoreographer\client`
   - Linux UNIX Sous les plateformes Linux® et UNIX® : *racine_installation*`/ProcessChoreographer/client`

3. Ajoutez les références EJB requises pour le descripteur de déploiement d'applications Web, le fichier `web.xml`.

```
<ejb-ref id="EjbRef_1">
  <ejb-ref-name>ejb/BusinessProcessHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
  <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
</ejb-ref>
<ejb-ref id="EjbRef_2">
  <ejb-ref-name>ejb/HumanTaskManagerEJB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.ibm.task.api.HumanTaskManagerHome</home>
  <remote>com.ibm.task.api.HumanTaskManager</remote>
</ejb-ref>
<ejb-local-ref id="EjbLocalRef_1">
  <ejb-ref-name>ejb/LocalBusinessProcessHome</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.bpe.api.LocalBusinessFlowManagerHome</local-home>
  <local>com.ibm.bpe.api.LocalBusinessFlowManager</local>
</ejb-local-ref>
<ejb-local-ref id="EjbLocalRef_2">
  <ejb-ref-name>ejb/LocalHumanTaskManagerEJB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>com.ibm.task.api.LocalHumanTaskManagerHome</local-home>
  <local>com.ibm.task.api.LocalHumanTaskManager</local>
</ejb-local-ref>
```

4. Ajoutez les composants JSF de Business Process Choreographer Explorer à l'application JSF.

   a. Ajoutez les références de bibliothèque de balises requises pour les applications dans les fichiers JavaServer Pages (JSP). En généralement, les ressources requises sont les bibliothèques de balises JSF et HTML et la bibliothèque de balises requise par les composants JSF.

      - `<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>`
      - `<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>`
      - `<%@ taglib uri="http://com.ibm.bpe.jsf/taglib" prefix="bpe" %>`

   b. Ajoutez une balise `<f:view>` au corps de la page JSP et une balise `<h:form>` à la balise `<f:view>`.

c. Ajoutez les composants JSF aux fichiers JSP.

   Selon votre application, ajoutez les composants List, Details, CommandBar ou Message aux fichiers JSP. Vous pouvez ajouter plusieurs instances à chaque composant.

d. Configurez les beans gérés dans le fichier de configuration JSF.

   Le fichier de configuration par défaut est `faces-config.xml`. Ce fichier réside dans le répertoire `WEB-INF` de l'application Web.

   Selon le composant que vous ajoutez à votre fichier JSP, vous devez également ajouter les références à la requête et aux objets d'encapsulation au fichier de configuration JSF. Pour s'assurer d'un traitement correct des erreurs, vous devez également définir un bean d'erreur et une cible de navigation pour la page d'erreur dans le fichier de configuration JSF. Utilisez `BPCError` comme nom pour le bean d'erreur et `error` comme nom pour la cible de navigation de la page d'erreur.

```
<faces-config>
...
<managed-bean>
  <managed-bean-name>BPCError</managed-bean-name>
  <managed-bean-class>com.ibm.bpc.clientcore.util.ErrorBeanImpl
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

...
<navigation-rule>
...
<navigation-case>
<description>
Page générale des erreurs.
</description>
<from-outcome>error</from-outcome>
<to-view-id>/Error.jsp</to-view-id>
</navigation-case>
...
</navigation-rule>
</faces-config>
```

   Lorsque des situations d'erreur entraînent le déclenchement de la page d'erreur, l'exception est définie au niveau du bean d'erreur.

e. Implémentez le code personnalisé requis pour la prise en charge des composants JSF.

5. Déployez l'application.

   Si vous déployez l'application dans un environnement de déploiement réseau, modifiez les noms JNDI (Java Naming and Directory Interface) des ressources cible avec des valeurs permettant de trouver les API de Business Flow Manager et Human Task Manager dans votre cellule.

   - Si vos conteneurs de processus métier sont configurés sur un autre serveur au sein de la même cellule gérée, les noms se présentent de la manière suivante :

     cellule/noeuds/*nomnoeud*/serveurs/*nomserveur*/com/ibm/bpe/api/BusinessManagerHome
     cellule/noeuds/*nomnoeud*/serveurs/*nomserveur*/com/ibm/task/api/HumanTaskManagerHome

   - Si vos conteneurs de processus métier sont configurés sur un serveur au sein de la même cellule, les noms se présentent de la manière suivante :

     cellule/clusters/*nomcluster*/com/ibm/bpe/api/BusinessFlowManagerHome
     cellule/clusters/*nomcluster*/com/ibm/task/api/HumanTaskManagerHome

   Mappez les références EJB avec les noms JNDI ou ajoutez manuellement les références au fichier `ibm-web-bnd.xmi`.

Le tableau suivant dresse la liste des liaisons de référence et leurs mappages par défaut.

Tableau 70. Mappage des liaisons de référence aux noms JNDI

Liaison de référence	Nom JNDI	Commentaires
ejb/BusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	Bean session distant
ejb/LocalBusinessProcessHome	com/ibm/bpe/api/BusinessFlowManagerHome	Bean session local
ejb/HumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	Bean session distant
ejb/LocalHumanTaskManagerEJB	com/ibm/task/api/HumanTaskManagerHome	Bean session local

task_results

Votre application Web déployée contient les fonctionnalités fournies par les composants de Business Process Choreographer Explorer.

task_postreq

Si vous utilisez des JSP personnalisés pour les messages de processus et de tâche, vous devez mapper les modules web qui sont utilisés pour déployer les JSP avec les mêmes serveurs que ceux avec lesquels est mappé le client JSF personnalisé.

Concepts associés

«Composants Exemples de Business Process Choreographer Explorer»
Les composants Business Process Choreographer Explorer constituent un ensemble d'éléments réutilisables configurables basés sur la technologie JavaServer Faces (JSF). Vous pouvez imbriquer ces éléments dans des applications Web. Les applications Web peuvent alors accéder à des applications de processus métier et de tâches manuelles installées.

«Traitement des erreurs dans les composants JSF», à la page 535
Les composants JavaServer Faces (JSF) exploitent un bean géré prédéfini, BPCError, pour le traitement des erreurs. Lorsque des situations d'erreur entraînent le déclenchement de la page d'erreur, l'exception est définie au niveau du bean d'erreur.

Tâches associées

«Ajout du composant List à une application JSF», à la page 536
Le composant List de Business Process Choreographer Explorer permet d'afficher une liste d'objets de modèle client tel qu'une liste d'instances de processus métier ou une instance de tâche.

«Ajout du composant Details à une application JSF», à la page 543
Le composant Details de Business Process Choreographer Explorer permet d'afficher les propriétés de tâches, de tâches élémentaires, d'instances de processus et de modèles de processus.

«Ajout du composant CommandBar à une application JSF», à la page 546
Utilisez le composant CommandBar de Business Process Choreographer Explorer pour permettre l'affichage d'une barre comportant des boutons de commande. Ces boutons représentent des commandes opérant dans une vue détails d'un objet ou des objets sélectionnés d'une liste.

«Ajout du composant Message à une application JSF», à la page 551
Le composant Message de Business Process Choreographer Explorer permet d'afficher des objets de données et des types de primitive dans une application JavaServer Faces (JSF).

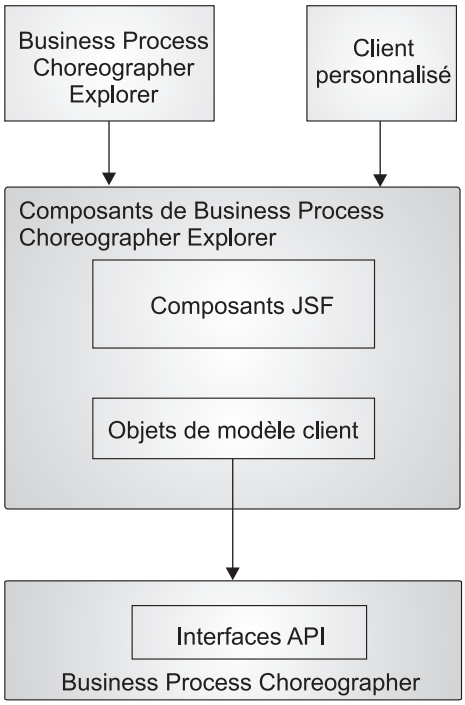
Référence associée

«Convertisseurs et intitulés par défaut d'objets de modèle client», à la page 536
Les objets de modèle client implémentent les interfaces correspondantes de l'API de Business Process Choreographer.

Composants Exemples de Business Process Choreographer Explorer

Les composants Business Process Choreographer Explorer constituent un ensemble d'éléments réutilisables configurables basés sur la technologie JavaServer Faces (JSF). Vous pouvez imbriquer ces éléments dans des applications Web. Les applications Web peuvent alors accéder à des applications de processus métier et de tâches manuelles installées.

Les composants consistent en un ensemble de composants JSF et un ensemble d'objets modèle client. La relation entre les composants et Business Process Choreographer, Business Process Choreographer Explorer et d'autres clients personnalisés est représentée dans la figure suivante.



Composants JSF

Les composants de Business Process Choreographer Explorer comprennent les composants JSF suivants. Ces composants JSF sont insérés dans les fichiers JavaServer Pages (JSP) lorsque vous générez des applications Web de gestion des processus métier et tâches manuelles.

- Composant List

  Le composant List affiche dans un tableau, une liste d'objets d'application tels que des tâches, des activités, des instances de processus, des modèles de processus, des éléments de travail ou des escalades. Ce composant possède un gestionnaire de liste associé.
- Composant Details

  Le composant Details permet d'afficher les propriétés de tâches, d'éléments de travail, d'instances de processus et modèles de processus. Ce composant possède un gestionnaire de détails associé.
- Composant CommandBar

  Le composant CommandBar permet d'afficher une barre avec boutons de commande. Ces boutons représentent des commandes qui agissent sur l'objet dans une vue détails ou les objets sélectionnés d'une liste. Ces objets sont fournis par un gestionnaire de listes ou un gestionnaire de détails.
- Composant Message

  Le composant Message affiche un message pouvant contenir un objet SDO (Service Data Object) ou un type simple.

Objets de modèle client

Les objets de modèle client sont utilisés avec les composants JSF. Les objets implémentent certaines interfaces de l'API de Business Process Choreographer sous-jacent et encapsule l'objet d'origine. Les objets de modèle client fournissent un support multilingue pour les libellés et les convertisseurs de certaines propriétés.

Traitement des erreurs dans les composants JSF

Les composants JavaServer Faces (JSF) exploitent un bean géré prédéfini, BPCError, pour le traitement des erreurs. Lorsque des situations d'erreur entraînent le déclenchement de la page d'erreur, l'exception est définie au niveau du bean d'erreur.

Ce bean met en oeuvre l'interface com.ibm.bpc.clientcore.util.ErrorBean. L'affichage de la page d'erreur a lieu dans les cas suivants :

- Lorsqu'une erreur se produit durant l'exécution d'une requête définie pour un gestionnaire de listes, et que cette erreur est générée en tant qu'erreur ClientException par la méthode execute d'une commande
- Lorsqu'une erreur ClientException est émise par la méthode execute d'une commande et qu'il ne s'agit pas d'une erreur ErrorsInCommandException, ou qu'elle ne met pas en oeuvre l'interface CommandBarMessage
- Si un message d'erreur est affiché dans le composant et que vous suivez l'hyperlien lié au message

Une mise en oeuvre par défaut de l'interface com.ibm.bpc.clientcore.util.ErrorBeanImpl est disponible.

L'interface est définie comme suit :

```
public interface ErrorBean {

    public void setException(Exception ex);

    /*
     * Cette méthode d'accès set permet de transmettre l'environnement
     * local et l'exception. Ainsi, les méthodes getExceptionMessage
     * peuvent renvoyer des chaînes localisées
     *
     */
    public void setException(Exception ex, Locale locale);

    public Exception getException();
    public String getStack();
    public String getNestedExceptionMessage();
    public String getNestedExceptionStack();
    public String getRootExceptionMessage();
    public String getRootExceptionStack();

    /*
     * Cette méthode renvoie le message d'exception
     * concaténé de façon récursive avec les messages de
     * toutes les exceptions imbriquées.
     */
    public String getAllExceptionMessages();

    /*
     * Cette méthode renvoie la pile d'exceptions
     * concaténée de façon récursive avec les piles
     * toutes les exceptions imbriquées.
     */
    public String getAllExceptionStacks();
}
```

Concepts associés

«Traitement des erreurs dans le composant List», à la page 541
Lorsque vous utilisez le composant List pour afficher des listes dans votre application JSF, vous pouvez tirer parti des fonctions de traitement d'erreurs fournies par la classe com.ibm.bpe.jsf.handler.BPCListHandler.

Convertisseurs et intitulés par défaut d'objets de modèle client

Les objets de modèle client implémentent les interfaces correspondantes de l'API de Business Process Choreographer.

Les composants List et Details fonctionnent sur tout type de bean. Vous pouvez afficher toutes les propriétés d'un bean. Toutefois, si vous voulez définir les convertisseurs et les intitulés utilisés pour les propriétés d'un bean, vous devez utiliser soit la balise `column` du composant List, soit la balise `property` du composant Details. Au lieu de définir les convertisseurs et les intitulés, vous pouvez définir des convertisseurs et des intitulés par défaut pour les propriétés en définissant les méthodes statiques suivantes. Vous pouvez définir les méthodes statiques suivantes :

```
static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
       getConverter(String property);
```

Le tableau suivant répertorie les objets de modèle client qui implémentent les classes d'API Business Flow Manager et Human Task Manager et fournissent les intitulés et le convertisseur par défaut pour leurs propriétés. Cet encapsulage des interfaces fournit des intitulés sensibles et des convertisseurs pour un ensemble de propriétés. Le tableau suivant répertorie les correspondances entre les interfaces de Business Process Choreographer et les objets de modèle client.

Tableau 71. Mappage d'interfaces de Business Process Choreographer avec des objets de modèle client

Interface de Business Process Choreographer	Classe d'objet de modèle client
com.ibm.bpe.api.ActivityInstanceData	com.ibm.bpe.clientmodel.bean.ActivityInstanceBean
com.ibm.bpe.api.ActivityServiceTemplateData	com.ibm.bpe.clientmodel.bean.ActivityServiceTemplateBean
com.ibm.bpe.api.ProcessInstanceData	com.ibm.bpe.clientmodel.bean.ProcessInstanceBean
com.ibm.bpe.api.ProcessTemplateData	com.ibm.bpe.clientmodel.bean.ProcessTemplateBean
com.ibm.task.api.Escalation	com.ibm.task.clientmodel.bean.EscalationBean
com.ibm.task.api.Task	com.ibm.task.clientmodel.bean.TaskInstanceBean
com.ibm.task.api.TaskTemplate	com.ibm.task.clientmodel.bean.TaskTemplateBean

Ajout du composant List à une application JSF

Le composant List de Business Process Choreographer Explorer permet d'afficher une liste d'objets de modèle client tel qu'une liste d'instances de processus métier ou une instance de tâche.

task_procedure

1. Ajoutez le composant List au fichier JavaServer Pages (JSP).

   Ajoutez la balise `bpe:list>` à la balise `h:form`. La balise `bpe:list>` doit contenir un attribut de modèle. Ajoutez des balises `bpe:column` à la balise `bpe:list` pour ajouter les propriétés des objets qui doivent figurer à chaque ligne de la liste.

L'exemple suivant illustre l'ajout d'un composant List afin d'afficher des instances de tâche.

```
<h:form>

   <bpe:list model="#{TaskPool}">
      <bpe:column name="name" action="taskInstanceDetails" />
      <bpe:column name="state" />
      <bpe:column name="kind" />
      <bpe:column name="owner" />
      <bpe:column name="originator" />
   </bpe:list>

</h:form>
```

L'attribut de modèle fait référence à un bean géré, TaskPool. Le bean géré fournit la liste d'objets Java traités par itération, puis affichés dans des lignes individuelles.

2. Configurez le bean géré référencé par la balise `bpe:list`.

   Pour le composant List, ce bean géré doit être une instance de la classe com.ibm.bpe.jsf.handler.BPCListHandler.

   L'exemple suivant illustre l'ajout d'un bean géré TaskPool au fichier de configuration.

```
<managed-bean>
<managed-bean-name>TaskPool</managed-bean-name>
<managed-bean-class>com.ibm.bpe.jsf.handler.BPCListHandler</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
   <managed-property>
      <property-name>query</property-name>
      <value>#{TaskPoolQuery}</value>
   </managed-property>
   <managed-property>
      <property-name>type</property-name>
      <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
   </managed-property>
</managed-bean>

<managed-bean>
<managed-bean-name>TaskPoolQuery</managed-bean-name>
<managed-bean-class>sample.TaskPoolQuery</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
   <managed-property>
     <property-name>type</property-name>
     <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
   </managed-property>
</managed-bean>

<managed-bean>
<managed-bean-name>htmConnection</managed-bean-name>
<managed-bean-class>com.ibm.task.clientmodel.HTMConnection</managed-bean-class>
<managed-bean-scope>application</managed-bean-scope>
   <managed-property>
      <property-name>jndiName</property-name>
      <value>java:comp/env/ejb/LocalHumanTaskManagerEJB</value>
   </managed-property>
</managed-bean>
```

L'exemple indique que TaskPool possède deux propriétés configurables : `query` et `type`. La valeur de la propriété query fait référence à un autre bean géré, TaskPoolQuery. La valeur de la propriété type indique la classe de bean dont les propriétés s'affichent dans les colonnes de la liste affichée. L'instance de requête associée possède également un type de propriété. Si un type de propriété est indiqué, il doit être identique au type indiqué pour le gestionnaire de liste.

Vous pouvez ajouter n'importe quel type de logique de requête à l'application JSF à partir du moment où le résultat de la requête peut être représenté sous forme de liste de beans fortement typés. Par exemple, la requête TaskPoolQuery est implémentée à l'aide d'une liste d'objets com.ibm.task.clientmodel.bean.TaskInstanceBean.

3. Ajoutez le code personnalisé du bean géré figurant en référence dans le gestionnaire de liste.

   L'exemple suivant illustre l'ajout de code personnalisé du bean géré TaskPool.

```
public class TaskPoolQuery implements Query {

  public List execute throws ClientException {

    // Rechercher dans le fichier faces-config le bean géré "htmConnection".
    //
    FacesContext ctx = FacesContext.getCurrentInstance();
    Application  app = ctx.getApplication();
    ValueBinding htmVb = app.createValueBinding("#{htmConnection}");
    htmConnection = (HTMConnection) htmVb.getValue(ctx);
    HumanTaskManagerService taskService =
         htmConnection.getHumanTaskManagerService();

    // Appel de la méthode de requête effective sur le service Human Task Manager.
    //
    // Ajouter à l'instruction de sélection les colonnes de base de données
    // de toutes les propriétés à afficher dans la liste
    //
    QueryResultSet queryResult = taskService.query(
     "DISTINCT TASK.TKIID, TASK.NAME, TASK.KIND, TASK.STATE, TASK.TYPE,"
      + "TASK.STARTER, TASK.OWNER, TASK.STARTED, TASK.ACTIVATED, TASK.DUE,
         TASK.EXPIRES, TASK.PRIORITY",
      "TASK.KIND IN(101,102,105) AND TASK.STATE IN(2)
        AND WORK_ITEM.REASON IN (1)",
      (Chaîne)null,
      (Integer)null,
      (TimeZone)null);
    List applicationObjects = transformToTaskList ( queryResult );
    return applicationObjects ;
  }


  private List transformToTaskList(QueryResultSet result) {

ArrayList array = null;
int entries = result.size();
array = new ArrayList( entries );

// Transformation de chaque ligne de QueryResultSet en bean d'instance de tâche.
 for (int i = 0; i < entries; i++) {
   result.next();
   array.add( new TaskInstanceBean( result, connection ));
 }
 return array ;
  }
}
```

   Le bean TaskPoolQuery interroge les propriétés des objets Java. Ce bean doit implémenter l'interface com.ibm.bpc.clientcore.Query. Quand il actualise son contenu, le gestionnaire de liste appelle la méthode execute de la requête. L'appel renvoie une liste d'objets Java. La méthode getType doit renvoyer le nom de classe des objets Java renvoyés.

task_results

Votre application JSF contient à présent une page JavaServer affichant les propriétés de la liste d'objets demandée : état, type, propriétaire et émetteur des tâches d'instance disponibles, par exemple.

Concepts associés

«Mode de traitement des listes»
Chaque instance du composant List est associée à une instance de la classe com.ibm.bpe.jsf.handler.BPCListHandler.

«Informations de fuseau horaire propres à l'utilisateur», à la page 540
Les composants JavaServer Faces (JSF) offrent un utilitaire de gestion des informations de fuseau horaire propre à l'utilisateur dans le composant List.

«Traitement des erreurs dans le composant List», à la page 541
Lorsque vous utilisez le composant List pour afficher des listes dans votre application JSF, vous pouvez tirer parti des fonctions de traitement d'erreurs fournies par la classe com.ibm.bpe.jsf.handler.BPCListHandler.

Référence associée

«Composant List : définitions de balises», à la page 542
Le composant List de Business Process Choreographer Explorer affiche dans un tableau, une liste d'objets d'application tels que des tâches, des activités, des instances de processus, des modèles de processus, des éléments de travail ou des escalades.

Mode de traitement des listes

Chaque instance du composant List est associée à une instance de la classe com.ibm.bpe.jsf.handler.BPCListHandler.

Le gestionnaire de listes effectue le suivi des éléments sélectionnés dans la liste associée et fournit un mécanisme de notification pour associer les entrées de liste aux pages de détails des différents types d'éléments. Le gestionnaire de listes est lié au composant List via l'attribut **model** contenu dans la balise `bpe:list`.

Le système de notification du gestionnaire de listes est mis en oeuvre via l'interface com.ibm.bpe.jsf.handler.ItemListener. Des implémentations de cette interface peuvent être enregistrées dans le fichier de configuration de votre application JSF (JavaServer Faces).

La notification est déclenchée en cas d'activation d'un lien dans la liste. Les liens de toutes les colonnes pour lesquelles l'attribut **action** est défini, s'affichent. La valeur de l'attribut **action** est soit une cible de navigation JSF, soit une méthode d'action JSF qui renvoie une cible de navigation JSF.

La classe BPCListHandler fournit également une méthode refreshList. Vous pouvez appliquer cette méthode à des liaisons de méthodes JSF afin de mettre en oeuvre un contrôle d'interface utilisateur visant à réexécuter la requête.

Mises en oeuvre de requêtes

Le gestionnaire de listes peut être utilisé pour afficher toutes sortes d'objets, ainsi que les propriétés de ces derniers. Le contenu de la liste affichée dépend de la liste des objets renvoyés par la mise en oeuvre de l'interface com.ibm.bpc.clientcore.Query configurée pour le gestionnaire de listes. Vous pouvez définir la requête par voie de programme via la méthode setQuery de la classe BPCListHandler, ou la configurer dans les fichiers de configuration JSF de l'application.

L'exécution de requêtes peut concerner non seulement les API de Business Process Choreographer, mais également toute autre source d'informations accessible par le biais de votre application, telle qu'un système de gestion de contenus ou une base de données. La seule condition requise est que le résultat de la requête soit renvoyé sous forme d'une liste `java.util.List` contenant les objets de la méthode execute.

Le type des objets renvoyés doit garantir que les méthodes d'accès get appropriées sont disponibles pour toutes les propriétés affichées dans les colonnes de la liste faisant l'objet de la requête. Pour vous assurer que le type d'objet renvoyé correspond bien aux définitions de la liste, vous pouvez utiliser le nom de classe qualifié complet des objets renvoyés en tant que valeur de propriété du type concerné dans l'instance BPCListHandler définie par le fichier de configuration JSF. Ce nom peut être renvoyé dans l'appel getType de la mise en oeuvre de la requête. Lors de l'exécution, le gestionnaire de listes contrôle que les types d'objet sont bien conformes aux définitions.

Pour créer un mappage entre des messages d'erreur et des entrées spécifiques d'une liste, les objets renvoyés par la requête doivent mettre en oeuvre une méthode comportant la signature `public Object getID()`.

Convertisseurs et intitulés par défaut

Les éléments renvoyés par une requête doivent être des beans et leurs classes doivent correspondre à la classe spécifiée comme le type dans la définition de la classe BPCListHandler ou de l'interface com.ibm.bpc.clientcore.Query. De plus, le composant List vérifie si la classe d'éléments ou une superclasse implémente les méthodes suivantes :

```
static public String getLabel(String property,Locale locale);
static public com.ibm.bpc.clientcore.converter.SimpleConverter
       getConverter(String property);
```

Si ces méthodes sont définies pour les beans, le composant List utilise l'intitulé comme intitulé par défaut pour la liste et SimpleConverter comme convertisseur par défaut pour la propriété. Vous pouvez remplacer ces paramètres par les attributs **label** et **converterID** de la balise `bpe:list`. Pour plus d'informations sur l'interface SimpleConverter et ColumnTag class, reportez-vous à la documentation Java.

Informations de fuseau horaire propres à l'utilisateur

Les composants JavaServer Faces (JSF) offrent un utilitaire de gestion des informations de fuseau horaire propre à l'utilisateur dans le composant List.

La classe BPCListHandler utilise l'interface com.ibm.bpc.clientcore.util.User pour obtenir des informations sur le fuseau horaire et l'environnement local de chaque utilisateur. Pour les besoins du composant List la mise en oeuvre de l'interface doit être configurée de sorte que **user** soit le nom du bean géré défini dans le fichier fichier de configuration JSF (JavaServer Faces). Si cette entrée est absente du fichier de configuration, la valeur renvoyée est celle du fuseau horaire dans lequel WebSphere Process Server est exécuté.

L'interface com.ibm.bpc.clientcore.util.User est définie comme suit :

```
public interface User {

    /**
     * Environnement local utilisé par le client de l'utilisateur.
     * @return Locale.
```

```
     */
    public Locale getLocale();
   /**
    * Fuseau horaire utilisé par le client de l'utilisateur.
    * @return TimeZone.
    */
    public TimeZone getTimeZone();

   /**
    * Nom de l'utilisateur.
    * @return nom de l'utilisateur.
    */
    public String getName();
}
```

Traitement des erreurs dans le composant List

Lorsque vous utilisez le composant List pour afficher des listes dans votre application JSF, vous pouvez tirer parti des fonctions de traitement d'erreurs fournies par la classe com.ibm.bpe.jsf.handler.BPCListHandler.

Erreurs se produisant lors de l'exécution de requêtes ou de commandes

Si une erreur se produit lors de l'exécution d'une requête, la classe BPCListHandler fait une distinction entre les erreurs dues à des droits d'accès insuffisants et les autres exceptions. Pour intercepter les erreurs dues à des droits d'accès insuffisants, le paramètre **rootCause** de l'exception ClientException lancée par la méthode execute de la requête doit être une exception de type com.ibm.bpe.api.EngineNotAuthorizedException ou com.ibm.task.api.NotAuthorizedException. Le composant List affiche le message d'erreur à la place du résultat de la requête.

Si l'erreur n'est pas provoquée par des droits d'accès insuffisants, la classe BPCListHandler transmet l'objet exception à la mise en oeuvre d'interface com.ibm.bpc.clientcore.util.ErrorBean qui est définie par la clé BPCError dans le fichier de configuration de l'application JSF. Une fois l'exception définie, la cible de navigation de l'erreur est appelée.

Erreurs se produisant lors du traitement d'entités affichées dans une liste

La classe BPCListHandler met en oeuvre l'interface com.ibm.bpe.jsf.handler.ErrorHandler. Vous pouvez fournir des informations sur ces erreurs via le paramètre de mappage de type java.util.Map inclus dans la méthode setErrors. Dans cette mappe, des identifiants sont associés à des clés et des exceptions sont associées à des valeurs. Les identifiants doivent obligatoirement être les valeurs renvoyées par la méthode getID de l'objet ayant provoqué l'erreur. Si la mappe est définie et qu'un ID correspond à l'une des entités de la liste, le gestionnaire de listes ajoute automatiquement à la liste une colonne contenant le message d'erreur.

Pour éviter que la liste ne contienne des messages d'erreur périmés, réinitialisez la mappe d'erreurs. La mappe est initialisée automatiquement dans les cas suivants :

- La classe BPCListHandler de la méthode refreshList est appelée.
- Une nouvelle requête est envoyée à la classe BPCListHandler.
- Le composant CommandBar est utilisé pour déclencher des actions concernant les entités contenues dans la liste. Le composant CommandBar utilise ce mécanisme comme méthode de traitement des erreurs.

Concepts associés

«Traitement des erreurs dans les composants JSF», à la page 535
Les composants JavaServer Faces (JSF) exploitent un bean géré prédéfini, BPCError, pour le traitement des erreurs. Lorsque des situations d'erreur entraînent le déclenchement de la page d'erreur, l'exception est définie au niveau du bean d'erreur.

Composant List : définitions de balises

Le composant List de Business Process Choreographer Explorer affiche dans un tableau, une liste d'objets d'application tels que des tâches, des activités, des instances de processus, des modèles de processus, des éléments de travail ou des escalades.

Le composant List comprend deux balises de composant JSF : `bpe:list` et `bpe:column`. La balise `bpe:column` est un sous-élément de la balise `bpe:list`.

Classe de composants

com.ibm.bpe.jsf.component.ListComponent

Syntaxe exemple

```
<bpe:list model="#{ProcessTemplateList}">
          rows="20"
          styleClass="list"
          headerStyleClass="listHeader"
          rowClasses="normal">

    <bpe:column name="name" action="processTemplateDetails"/>
    <bpe:column name="validFromTime"/>
    <bpe:column name="executionMode" label="Execution mode"/>
    <bpe:column name="state" converterID="my.state.converter"/>
    <bpe:column name="autoDelete"/>
    <bpe:column name="description"/>

</bpe:list>
```

Attributs de balise

Le corps de la balise `bpe:list` ne peut contenir que des balises `bpe:column`. Quand la table s'affiche, le composant List effectue une itération sur sa liste d'objets d'application et affiche toutes les colonnes de chaque objet.

Tableau 72. Attributs bpe:list

Attribut	Obligatoire	Description
buttonStyleClass	non	Classe de styles CSS pour l'affichage des boutons dans la zone de pied de page.
cellStyleClass	non	Classe de styles CSS pour l'affichage de cellules de tableau.
checkbox	non	Détermine si la case à cocher de sélection multiple est affichée. L'attribut possède la valeur `true` ou `false`. Si la valeur est définie sur `true`, la colonne de case à cocher est affichée.
headerStyleClass	non	Classe de styles CSS pour l'affichage de l'entête de tableau.

Tableau 72. Attributs bpe:list (suite)

Attribut	Obligatoire	Description
model	oui	Liaison de valeur d'un bean géré de la classe com.ibm.bpe.jsf.handler.BPCListHandler.
rows	non	Nombre de lignes affichées par page. Si le nombre d'éléments est supérieur au nombre de lignes, des boutons de pagination s'affichent à la fin du tableau. Les expressions de valeur ne sont pas prises en charge pour cet attribut.
rowClasses	non	Classe de styles CSS pour l'affichage des lignes du tableau.
selectAll	non	Si cet attribut est défini à `true`, tous les éléments de la liste sont sélectionnés par défaut.
styleClass	non	Classe de styles CSS pour l'affichage du tableau contenant les titres, les lignes et les boutons de pagination.

Tableau 73. Attributs bpe:column

Attribut	Obligatoire	Description
action	non	Si cet attribut est indiqué, un lien s'affiche dans cette colonne. Quand vous cliquez sur ce lien, cela provoque le déclenchement d'une méthode d'action JavaServer Faces ou de la cible de navigation Faces. Une méthode d'action JavaServer Faces possède la signature : String method().
converterID	non	L'identificateur du convertisseur Faces utilisé pour convertir la valeur de la propriété. Si cet attribut n'est pas défini, l'identificateur du convertisseur Faces fourni par le modèle pour cette propriété est utilisé.
label	non	Expression littérale ou de liaison de valeur utilisée en tant qu'intitulé de l'en-tête de la colonne ou de la cellule de la ligne d'en-tête de table. Si cet attribut n'est pas défini, l'intitulé fourni par le modèle pour cette propriété est utilisé.
name	oui	Nom de la propriété qui est affichée dans cette colonne.

Ajout du composant Details à une application JSF

Le composant Details de Business Process Choreographer Explorer permet d'afficher les propriétés de tâches, de tâches élémentaires, d'instances de processus et de modèles de processus.

task_procedure

1. Ajoutez le composant Details au fichier JavaServer Pages (JSP).

Ajoutez la balise bpe:details à la balise <h:form>. La balise bpe:details doit contenir un attribut de **modèle**. Vous pouvez ajouter des propriétés au composant Details à l'aide de la balise bpe:property.

L'exemple suivant illustre l'ajout d'un composant Details afin d'afficher quelques-unes des propriétés d'une instance de tâche.

```
<h:form>

   <bpe:details model="#{TaskInstanceDetails}">
      <bpe:property name="displayName" />
      <bpe:property name="owner" />
      <bpe:property name="kind" />
      <bpe:property name="state" />
      <bpe:property name="escalated" />
      <bpe:property name="suspended" />
      <bpe:property name="originator" />
      <bpe:property name="activationTime" />
      <bpe:property name="expirationTime" />
   </bpe:details>

</h:form>
```

L'attribut de **modèle** fait référence à un bean géré, TaskInstanceDetails. Le bean fournit les propriétés de l'objet Java.

2. Configurez le bean géré référencé par la balise bpe:details.

   Pour le composant Details, ce bean géré doit être une instance de la classe com.ibm.bpe.jsf.handler.BPCDetailsHandler. Cette classe de gestionnaire encapsule un objet Java et expose ses propriétés publiques au composant Details.

   L'exemple suivant illustre l'ajout d'un bean géré TaskInstanceDetails au fichier de configuration.

```
<managed-bean>
   <managed-bean-name>TaskInstanceDetails</managed-bean-name>
   <managed-bean-class>com.ibm.bpe.jsf.handler.BPCDetailsHandler</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
   <managed-property>
      <property-name>type</property-name>
      <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
   </managed-property>
</managed-bean>
```

   L'exemple montre que le bean TaskInstanceDetails bean possède une propriété type configurable. La valeur de la propriété type indique la classe de bean (com.ibm.task.clientmodel.bean.TaskInstanceBean) dont les propriétés s'affichent dans les lignes de détail générées. La classe de bean peut correspondre à n'importe quelle classe JavaBeans. Si le bean fournit des intitulés de conversion et de propriété par défaut, le convertisseur et l'intitulé sont utilisés pour le rendu de la même manière que le composant List.

task_results

Votre application JSF contient à présent une page JavaServer affichant les détails de l'objet spécifié (une instance de tâche, par exemple).

Référence associée

«Composant Details : définitions de balises»
Le composant Details de Business Process Choreographer Explorer permet d'afficher les propriétés de tâches, d'éléments de travail, d'instances de processus et modèles de processus.

Composant Details : définitions de balises

Le composant Details de Business Process Choreographer Explorer permet d'afficher les propriétés de tâches, d'éléments de travail, d'instances de processus et modèles de processus.

Le composant Details comprend deux balises de composant JSF : `bpe:details` et `bpe:property`. La balise `bpe:property` est un sous-élément de la balise `bpe:details`.

Classe de composants

com.ibm.bpe.jsf.component.DetailsComponent

Syntaxe exemple

```
<bpe:details model="#{MyActivityDetails}">
    <bpe:property name="name"/>
    <bpe:property name="owner"/>
    <bpe:property name="activated"/>
</bpe:details>
```

```
<bpe:details model="#{MyActivityDetails}" style="style" styleClass="cssStyle">
             style="style"
             styleClass="cssStyle"
</bpe:details>
```

Attributs de balise

Les balises `bpe:property` permettent d'indiquer à la fois le sous-ensemble d'attributs affichés et l'ordre d'affichage de ces attributs. Si la balise détails ne contient pas de balise d'attribut, elle affiche tous les attributs disponibles de l'objet modèle.

Tableau 74. Attributs bpe:details

Attribut	Obligatoire	Description
columnClasses	non	Liste des classes de style CSS séparées par des virgules et utilisées pour l'affichage de colonnes.
id	non	ID du composant JavaServer Faces.
model	oui	Liaison de valeur d'un bean géré de la classe com.ibm.bpe.jsf.handler.BPCDetailsHandler.
rowClasses	non	Liste des classes de style CSS séparées par des virgules et utilisées pour l'affichage des lignes.
styleClass	non	Classe CSS utilisée pour l'affichage de l'élément HTML.

Tableau 75. Attributs bpe:property

Attribut	Obligatoire	Description
converterID	non	Identificateur utilisé pour l'enregistrement du convertisseur dans le fichier de configuration JavaServer Faces (JSF).
label	non	Libellé de la propriété. Si cet attribut n'est pas défini, un libellé par défaut est fourni par la classe de modèle client.
name	oui	Nom de la propriété à afficher. Ce nom doit correspondre à une propriété nommée définie dans la classe de modèle client correspondant.

Ajout du composant CommandBar à une application JSF

Utilisez le composant CommandBar de Business Process Choreographer Explorer pour permettre l'affichage d'une barre comportant des boutons de commande. Ces boutons représentent des commandes opérant dans une vue détails d'un objet ou des objets sélectionnés d'une liste.

task_context

Quand l'utilisateur clique sur un bouton dans l'interface, la commande correspondante est exécutée sur les objets sélectionnés. Vous pouvez ajouter et étendre le composant CommandBar dans votre application JSF (JavaServer Faces).

task_procedure

1. Ajoutez le composant CommandBar au fichier JavaServer Pages (JSP).

   Ajoutez la balise `bpe:commandbar` à la balise `<h:form>`. La balise `bpe:commandbar` doit contenir un attribut de modèle.

   L'exemple suivant illustre l'ajout d'un composant CommandBar, ce dernier fournissant des commandes de régénération et de réclamation pour une liste d'instances de tâches.

   ```
   <h:form>

      <bpe:commandbar model="#{TaskInstanceList}">
         <bpe:command commandID="Refresh" >
                      action="#{TaskInstanceList.refreshList}"
                      label="Refresh"/>

         <bpe:command commandID="MyClaimCommand" >
                      label="Claim" >
                      commandClass="<customcode>"/>
      </bpe:commandbar>

   </h:form>
   ```

   L'attribut **model** fait référence à un bean géré. Ce bean doit implémenter l'interface ItemProvider et fournir les objets Java sélectionnés. Le composant CommandBar est généralement utilisé soit avec le composant List, soit avec le composant Details dans le même fichier JSP. En général, le modèle spécifié dans la balise correspond à celui qui est indiqué dans le composant List ou Details sur la même page. Pour un composant List, la commande agit donc sur les éléments sélectionnés dans la liste.

   Dans cet exemple, l'attribut **model** fait référence au bean géré TaskInstanceList. Ce bean fournit les objets sélectionnés dans la liste des instances de tâches. Il

doit implémenter l'interface ItemProvider. Cette interface est implémentée par les classes BPCListHandler et BPCDetailsHandler.

2. OptionalColonSymbol Configurez le bean géré référencé par la balise `bpe:commandbar`.

   Si l'attribut **model** de CommandBar fait référence à un bean géré qui est déjà configuré, par exemple dans le cas d'une liste ou d'un gestionnaire de détails, aucune configuration complémentaire n'est requise. Si vous n'utilisez ni la classe BPCListHandler, ni la classe BPCDetailsHandler pour le modèle, vous devez faire référence à un autre objet comportant une classe qui implémente l'interface ItemProvider.

3. Ajoutez le code implémentant les commandes personnalisés vers l'application JSF.

   Le fragment de code ci-dessous montre comment écrire une classe de commandes qui implémente l'interface Command. Cette classe de commandes (MyClaimCommand) est désignée par la balise `bpe:command` dans le fichier JSP.

```
public class MyClaimCommand implements Command {

  public String execute(List selectedObjects) throws ClientException {
     if( selectedObjects != null && selectedObjects.size() > 0 ) {
        try {
         // Déterminer HumanTaskManagerService à partir d'un bean HTMConnection.
         // Configurer le bean dans le fichier faces-config.xml pour faciliter
         // l'accès à l'application JSF.
         FacesContext ctx = FacesContext.getCurrentInstance();
         ValueBinding vb =
           ctx.getApplication().createValueBinding("{htmConnection}");
         HTMConnection htmConnection = (HTMConnection) htmVB.getValue(ctx);
         HumanTaskManagerService htm =
            htmConnection.getHumanTaskManagerService();

         Iterator iter = selectedObjects.iterator() ;
         while( iter.hasNext() ) {
           try {
                TaskInstanceBean task = (TaskInstanceBean) iter.next() ;
                TKIID tiid = task.getID() ;

                htm.claim( tiid ) ;
                task.setState( new Integer(TaskInstanceBean.STATE_CLAIMED ) ) ;

           }
           catch( Exception e ) {
             ;   // Erreur lors de l'itération ou réclamation d'une instance de tâche.
                  // Ignorer pour mieux comprendre l'exemple.
           }
         }
       }
       catch( Exception e ) {
         ;  // Erreur de configuration ou de communication.
           // Ignorer pour mieux comprendre l'exemple.
       }
     }
     return null;
  }

 // Implémentations par défaut
 public boolean isMultiSelectEnabled() { return false; }
 public boolean[] isApplicable(List itemsOnList) {return null; }
 public void setContext(Object targetModel) {; // Non utilisé ici }
}
```

   La commande est traitée ainsi :

a. Une commande est appelée quand un utilisateur clique sur le bouton correspondant dans la barre de commandes. Le composant CommandBar extrait les éléments sélectionnés depuis le fournisseur d'éléments indiqué dans l'attribut **model** et transmet la liste d'objets sélectionnés à la méthode execute de l'instance commandClass.
b. OptionalColonSymbol L'attribut **commandClass** fait référence à une implémentation de commande personnalisée mettant en oeuvre l'interface Command. Cela signifie que la commande doit implémenter la méthode public String execute(List selectedObjects) throws ClientException. Elle renvoie le résultat permettant de déterminer la prochaine règle de navigation de l'application JSF.
c. OptionalColonSymbol Après l'exécution de la commande, le composant CommandBar évalue l'attribut **action**. L'attribut **action** peut être une chaîne statique ou une liaison de méthode vers une méthode d'action ayant la signature `public String Method()`. L'attribut **action** permet de remplacer le résultat d'une classe de commandes ou d'indiquer explicitement un résultat pour les règles de navigation. L'attribut **action** n'est pas traité si la commande génère une exception autre que ErrorsInCommandException.
d. Si aucune classe de commandes n'est spécifiée pour l'attribut **commandClass**, l'action est immédiatement appelée. Par exemple, pour la commande refresh utilisée dans l'exemple, c'est l'expression de valeur JSF `#{TaskInstanceList.refreshList}` qui est appelée au lieu d'une commande.

task_results

Votre application JSF contient à présent une page JSP implémentant une barre de commandes personnalisée.

Concepts associés

«Mode de traitement des commandes»
Utilisez le composant CommandBar pour intégrer des boutons d'action à votre application. Le composant crée les boutons qui correspondent aux actions dans l'interface utilisateur et traite les événements générés lors du clic sur un bouton.

Référence associée

«Composant CommandBar : définitions de balises», à la page 549
Le composant CommandBar de Business Process Choreographer Explorer permet d'afficher une barre comportant des boutons de commande. Ces boutons agissent sur l'objet dans une vue détails ou les objets sélectionnés d'une liste.

Mode de traitement des commandes

Utilisez le composant CommandBar pour intégrer des boutons d'action à votre application. Le composant crée les boutons qui correspondent aux actions dans l'interface utilisateur et traite les événements générés lors du clic sur un bouton.

Ces boutons déclenchent des fonctions agissant sur les objets renvoyés par une interface com.ibm.bpe.jsf.handler.ItemProvider tels que la classe BPCListHandler, ou encore la classe BPCDetailsHandler. Le composant CommandBar utilise le fournisseur d'éléments défini par la valeur de l'attribut **model** contenu dans la balise `bpe:commandbar`.

Lorsqu'un clic est effectué sur un bouton situé dans la section dédiée à la barre de commandes dans l'interface utilisateur de l'application, l'événement associé est traité comme suit par le composant CommandBar.

1. Le composant CommandBar identifie la mise en oeuvre de l'interface com.ibm.bpc.clientcore.Command spécifiée pour le bouton ayant généré l'événement.
2. Si le modèle associé au composant CommandBar met en oeuvre l'interface com.ibm.bpe.jsf.handler.ErrorHandler, la méthode clearErrorMap est appelée pour effacer les messages d'erreur consécutifs aux événements antérieurs.
3. La méthode getSelectedItems de l'interface ItemProvider est appelée. La liste des entités renvoyées est transmise à la méthode execute de la commande, puis cette dernière est appelée.
4. Le composant CommandBar détermine la cible de navigation JSF (JavaServer Faces). Si aucun attribut **action** n'est spécifié dans la balise `bpe:commandbar`, la cible de navigation est spécifiée par la valeur renvoyée de la méthode execute. Si l'attribut **action** est défini sur une liaison de méthode JSF, la chaîne renvoyée par la méthode est interprétée comme étant la cible de navigation. L'attribut **action** peut également spécifier une cible de navigation explicite.

Composant CommandBar : définitions de balises

Le composant CommandBar de Business Process Choreographer Explorer permet d'afficher une barre comportant des boutons de commande. Ces boutons agissent sur l'objet dans une vue détails ou les objets sélectionnés d'une liste.

Le composant CommandBar comprend deux balises de composant JSF : `bpe:commandbar` et `bpe:command`. La balise `bpe:command` est un sous-élément de la balise `bpe:commandbar`.

Classe de composants

com.ibm.bpe.jsf.component.CommandBarComponent

Syntaxe exemple

```
<bpe:commandbar model="#{TaskInstanceList}">

    <bpe:command
       commandID="Work on"
       label="Work on..."
       commandClass="com.ibm.bpc.explorer.command.WorkOnTaskCommand"
       context="#{TaskInstanceDetailsBean}"/>

    <bpe:command
       commandID="Cancel"
       label="Cancel"
       commandClass="com.ibm.task.clientmodel.command.CancelClaimTaskCommand"
       context="#{TaskInstanceList}"/>

</bpe:commandbar>
```

Attributs de balise

Tableau 76. Attributs bpe:commandbar

Attribut	Obligatoire	Description
buttonStyleClass	non	Classe de styles CSS pour l'affichage des boutons de la barre de commandes.
id	non	ID du composant JavaServer Faces.

Tableau 76. Attributs bpe:commandbar (suite)

Attribut	Obligatoire	Description
model	oui	Expression de liaison de valeur vers un bean géré implémentant une interface ItemProvider. Ce bean géré est généralement la classe com.ibm.bpe.jsf.handler.BPCListHandler ou la classe com.ibm.bpe.jsf.handler.BPCDetailsHandler utilisée par le composant List ou Details dans le même fichier JavaServer Pages (JSP) que le composant CommandBar.
styleClass	non	Classe de styles CSS pour l'affichage de la barre de commandes.

Tableau 77. Attributs bpe:command

Attribut	Obligatoire	Description
action	non	Méthode d'action JavaServer Faces ou cible de navigation Faces qui est déclenchée par le bouton de commande. La cible de navigation qui est renvoyée par l'action écrase toutes les autres règles de navigation. L'action est appelée lorsqu'une exception n'est pas émise ou lorsqu'une exception ErrorsInCommandException est émise par la commande.
commandClass	non	Le nom de la classe de commande. Une instance de la classe est créée par le composant CommandBar, puis elle est exécutée lorsque le bouton de commande est sélectionné.
commandID	oui	ID de la commande.
context	non	Un objet qui fournit du contexte pour les commandes qui sont spécifiées à l'aide de l'attribut **commandClass**. L'objet de contexte est extrait lors du premier accès à la barre de commandes.
immediate	non	Indique le moment du déclenchement de la commande. Si la valeur de cet attribut est définie sur `true`, la commande est déclenchée avant le traitement de l'entrée de la page. La valeur par défaut est `false`.
label	oui	Libellé du bouton affiché dans la barre de commandes.
rendu	non	Détermine si un bouton a été rendu. La valeur de l'attribut peut être une valeur booléenne ou une expression de valeur.
styleClass	non	Classe CSS utilisée pour l'affichage du bouton. Ce style se substitue au style de bouton défini pour la barre de commandes.

Ajout du composant Message à une application JSF

Le composant Message de Business Process Choreographer Explorer permet d'afficher des objets de données et des types de primitive dans une application JavaServer Faces (JSF).

task_context

Si le message est de type primitif, un libellé et un champ de saisie sont affichés. Si le type de message est un objet de données, le composant traverse l'objet et affiche les éléments à l'intérieur de l'objet.

task_procedure

1. Ajoutez le composant Message au fichier JavaServer Pages (JSP).

   Ajoutez la balise `bpe:form` à la balise `<h:form>`. La balise `bpe:form` doit contenir un attribut model.

   L'exemple suivant illustre l'ajout d'un composant Message.

   ```
   <h:form>

      <h:outputText value="Input Message" />
      <bpe:form model="#{MyHandler.inputMessage}" readOnly="true" />

      <h:outputText value="Output Message" />
      <bpe:form model="#{MyHandler.outputMessage}" />

   </h:form>
   ```

   L'attribut **model** du composant Message fait référence à un objet com.ibm.bpc.clientcore.MessageWrapper. Cet objet encapsuleur enveloppe un objet SDO (Service Data Object) ou une primitive de type Java, par exemple de type `int` ou `boolean`. Dans l'exemple, le message est fourni par une propriété du bean géré MyHandler.

2. Configurez le bean géré référencé par la balise `bpe:form`.

   L'exemple suivant illustre l'ajout d'un bean géré MyHandler au fichier de configuration.

   ```
   <managed-bean>
   <managed-bean-name>MyHandler</managed-bean-name>
   <managed-bean-class>com.ibm.bpe.sample.jsf.MyHandler</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>

      <managed-property>
         <property-name>type</property-name>
         <value>com.ibm.task.clientmodel.bean.TaskInstanceBean</value>
      </managed-property>

   </managed-bean>
   ```

3. Ajoutez du code personnalisé à l'application JSF.

   L'exemple suivant illustre l'implémentation de messages d'entrée et de sortie.

   ```
   public class MyHandler implements ItemListener {

     private TaskInstanceBean taskBean;
     private MessageWrapper inputMessage, outputMessage

     /* Listener method, e.g. when a task instance was selected in a list handler.
      * Ensure that the handler is registered in the faces-config.xml or manually.
      */
     public void itemChanged(Object item) {
       if( item instanceof TaskInstanceBean ) {
           taskBean = (TaskInstanceBean) item ;
       }
   ```

```
  }

  /* Get the input message wrapper
   */
  public MessageWrapper getInputMessage() {
    try{
        inputMessage = taskBean.getInputMessageWrapper() ;
    }
    catch( Exception e ) {
        ;       //...ignore errors for simplicity
    }
    return inputMessage;
  }

  /* Get the output message wrapper
   */
  public MessageWrapper getOutputMessage() {
    Extraction du message du bean. Si aucun message n'existe, créez-en
    // un si la tâche a été réclamée par l'utilisateur. Assurez-vous que
    // seuls les propriétaires (potentiels ou non) peuvent manipuler
le message de sortie.
    try{
        outputMessage = taskBean.getOutputMessageWrapper();
        if( outputMessage == null
         && taskBean.getState() == TaskInstanceBean.STATE_CLAIMED ) {
            HumanTaskManagerService htm = getHumanTaskManagerService();
            outputMessage = new MessageWrapperImpl();
            outputMessage.setMessage(
               htm.createOutputMessage( taskBean.getID() ).getObject()
            );
         }
    }
    catch( Exception e ) {
        ;       //...ignore errors for simplicity
    }
    return outputMessage
  }
}
```

Le bean géré MyHandler implémente l'interface com.ibm.jsf.handler.ItemListener pour permettre son enregistrement en tant qu'écouteur d'éléments du gestionnaires de listes. Quand l'utilisateur clique sur un élément dans la liste, le bean MyHandler est informé sur l'élément sélectionné via la méthode itemChanged(Object item). Le gestionnaire contrôle le type d'élément, puis stocke une référence à l'objet TaskInstanceBean associé. Pour utiliser cette interface, ajoutez une entrée dans la liste itemListener du gestionnaire de listes approprié, qui se trouve dans le fichier `faces-config.xml`.

Le bean MyHandler fournit les méthodes getInputMessage et getOutputMessage. Ces deux méthodes retournent un objet MessageWrapper. Les méthodes délèguent les appels du bean d'instance de tâche référencé. Si l'instance de tâche renvoie la valeur null, par exemple parce qu'un message n'est pas défini, le gestionnaire crée et stocke un nouveau message vide. Le composant Message affiche les messages fournis par le bean MyHandler.

task_results

Votre application JSF contient à présent une page JSP permettant d'afficher des objets de données et des types primitifs.

Référence associée

«Composant Message : définitions de balises»
Le composant Message de Business Process Choreographer Explorer affiche des objets `commonj.sdo.DataObject` et des types de primitive, tels que des entiers et des chaînes, dans une application JavaServer Faces (JSF).

Composant Message : définitions de balises

Le composant Message de Business Process Choreographer Explorer affiche des objets `commonj.sdo.DataObject` et des types de primitive, tels que des entiers et des chaînes, dans une application JavaServer Faces (JSF).

Le composant Message comprend la balise de composant JSF : `bpe:form`.

Classe de composants

com.ibm.bpe.jsf.component.MessageComponent

Syntaxe exemple

```
<bpe:form model="#{TaskInstanceDetailsBean.inputMessageWrapper}"
          simplification="true" readOnly="true"
          styleClass4table="messageData"
          styleClass4output="messageDataOutput">
</bpe:form>
```

Attributs de balise

Tableau 78. Attributs bpe:form

Attribut	Obligatoire	Description
id	non	ID du composant JavaServer Faces.
model	oui	Expression de liaison de valeur qui fait référence à un objet `commonj.sdo.DataObject` ou à un objet `com.ibm.bpc.clientcore.MessageWrapper`.
readOnly	non	Si cet attribut est réglé sur `true`, un formulaire s'affiche en lecture seule. Par défaut, cet attribut est réglé sur `false`.
simplification	non	Si cet attribut est réglé sur `true`, les propriétés contenant des types simples et ayant une cardinalité de 0 ou de 1 sont affichées. Par défaut, cet attribut est défini sur `true`.
style4validinput	non	Style CSS (feuille de styles en cascade) pour l'affichage de valeur d'entrée valide.
style4invalidinput	non	Style CSS pour l'affichage de valeur d'entrée incorrecte.
styleClass4invalidInput	non	Nom de classe de style CSS pour l'affichage de valeur d'entrée incorrecte.
styleClass4output	non	Nom de classe de styles CSS pour l'affichage d'éléments sortants.
styleClass4table	non	Nom de classe du style de tableau CSS pour l'affichage des tableaux affichés par le composant de message.
styleClass4validInput	non	Nom de classe de style CSS pour l'affichage de valeur d'entrée correcte.

Développement des pages JSP pour les messages de tâche et de processus

Business Process Choreographer Explorer fournit des formulaires d'entrée et de sortie par défaut pour afficher et saisir les données métier. Vous pouvez utiliser des pages JSP pour créer des formulaires d'entrée et de sortie définis par l'utilisateur.

task_context

Pour inclure des pages JSP (JavaServer Pages) définies par l'utilisateur dans le client Web, vous devez les indiquer lorsque vous modélisez une tâche manuelle dans WebSphere Integration Developer. Par exemple, vous pouvez fournir des pages JSP pour une tâche spécifique et pour les messages d'entrée et de sortie associés, ainsi que pour un rôle utilisateur spécifique ou pour tous les rôles utilisateur. Lors de l'exécution, les pages JSP définies par l'utilisateur sont incluses dans l'interface utilisateur pour afficher les données de sortie et collecter les données d'entrée.

Les formulaires personnalisés ne sont pas des pages Web autonomes ; il s'agit de fragments de code HTML que Business Process Choreographer Explorer intègre dans un formulaire HTML (par exemple, les fragments pour tous les libellés et les zones d'entrée d'un message).

Lorsqu'un utilisateur clique sur un bouton de la page contenant les formulaires personnalisés, les données d'entrée sont soumises et validées dans Business Process Choreographer Explorer. La validation dépend du type des propriétés fournies et des paramètres locaux utilisés dans le navigateur. Si les données d'entrée ne peuvent pas être validées, la même page s'affiche de nouveau et les informations relatives aux erreurs de validation sont fournies dans l'attribut de demande messageValidationErrors. Les informations sont fournies sous forme d'un plan qui mappe l'expression XPath (XML Path Expression) des propriétés non valides avec les exceptions de validation qui ont eu lieu.

Pour ajouter des formulaires personnalisés à Business Process Choreographer Explorer, exécutez les opérations suivantes à l'aide de WebSphere Integration Developer :

task_procedure

1. Créez les formulaires personnalisés.

   Les pages JSP définies par l'utilisateur pour les formulaires d'entrée et de sortie utilisés dans l'interface Web doivent accéder aux données de messages. Utilisez les fragments Java d'un JSP ou le langage d'exécution JSP pour accéder aux données du message. Les données contenues dans les formulaires sont accessibles via le contexte de requête.
2. Affectez les pages JSP à une tâche.

   Ouvrez la tâche manuelle dans l'éditeur de tâches manuelles. Dans les paramètres client, indiquez l'emplacement des pages JSP définies par l'utilisateur et le rôle auquel s'applique le formulaire personnalisé (par exemple, administrateur). Les paramètres client de Business Process Choreographer Explorer sont stockés dans le modèle de tâche. Lors de l'exécution, ces paramètres sont extraits avec le modèle de tâche.

3. Compressez les pages JSP définies par l'utilisateur dans une archive Web (fichier WAR).

   Vous pouvez inclure le fichier WAR dans le fichier EAR (Enterprise Archive) avec le module contenant les tâches ou déployer le fichier WAR séparément. Si les JSP sont déployés séparément, faites en sorte qu'ils soient disponibles sur le serveur où est déployé Business Process Choreographer Explorer ou le client défini par l'utilisateur.

   Si vous utilisez des JSP personnalisés pour les messages de processus et de tâche, vous devez mapper les modules web qui sont utilisés pour déployer les JSP avec les mêmes serveurs que ceux avec lesquels est mappé le client JSF personnalisé.

task_results

Les formulaires personnalisés s'affichent dans Business Process Choreographer Explorer lors de l'exécution.

Concepts associés

«Fragments JSP définis par l'utilisateur»
Les fragments JSP (JavaServer Pages) définis par l'utilisateur sont intégrés à une balise de formulaire HTML. Lors de l'exécution, Business Process Choreographer Explorer inclut ces fragments dans la page affichée.

Fragments JSP définis par l'utilisateur

Les fragments JSP (JavaServer Pages) définis par l'utilisateur sont intégrés à une balise de formulaire HTML. Lors de l'exécution, Business Process Choreographer Explorer inclut ces fragments dans la page affichée.

Le fragment JSP défini par l'utilisateur du message d'entrée est intégré avant le fragment JSP du message de sortie.

```
<html....>
   ...
   <form...>
     Message JSP d'entrée (affichage du message d'entrée de la tâche)

     Message JSP de sortie (affichage du message de sortie de la tâche)

   </form>
   ...
</html>
```

Les fragments JSP définis par l'utilisateur étant intégrés à une balise de formulaire HTML, vous pouvez ajouter des éléments d'entrée. Le nom de l'élément d'entrée doit correspondre à l'expression XPath (XML Path Language) de l'élément de données. Il est important de faire précéder de la valeur de préfixe fournie le nom de l'élément d'entrée :

```
<input id="address"
       type="text"
       name="${prefix}/selectPromotionalGiftResponse/address"
       value="${messageMap['/selectPromotionalGiftResponse/address"]}
       size="60"
       align="left" />
```

La valeur de préfixe est fournie sous forme d'attribut de demande. L'attribut garantit l'unicité du nom d'entrée dans le formulaire d'inclusion. Le préfixe est généré par Business Process Choreographer Explorer et ne doit pas être modifié :

```
String prefix = (String)request.getAttribute("prefix");
```

L'élément de préfixe est défini uniquement si le message peut être modifié dans le contexte spécifié. Les données de sortie peuvent s'afficher de différentes façons selon l'état de la tâche manuelle. Par exemple, si l'état de la tâche est Réclamé, les données de sortie peuvent être modifiées. Toutefois, si l'état de la tâche est Terminé, les données peuvent uniquement être affichées. Dans votre fragment JSP, vous pouvez vérifier si l'élément de préfixe existe et afficher le message en conséquence. L'instruction JSTL suivante montre comment vérifier si l'élément de préfixe est défini :

```
...
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
...
<c:choose>
  <c:when test="${not empty prefix}">
    <!--Read/write mode-->
  </c:when>
  <c:otherwise>
    <!--Read-only mode-->
  </c:otherwise>
</c:choose>
```

Création de plug-in pour personnaliser les fonctionnalités des tâches manuelles

Business Process Choreographer fournit une infrastructure permettant le traitement des événements qui surviennent lors du traitement des tâches manuelles. L'application des plug-in est également conçue pour vous permettre d'adapter les fonctionnalités à vos besoins. Vous pouvez utiliser les interfaces de fournisseur de services SPI (Service Provider Interfaces) afin de créer des plug-in personnalisés pour la gestion des événements et le post-traitement des requêtes de personnel.

task_context

Vous pouvez créer des plug-in pour des événements liés à des API de tâche manuelle et à des notifications d'escalade. Vous pouvez également créer un plug-in qui traite les résultats renvoyés par la résolution des utilisateurs. Vous pouvez par exemple, lors de pics périodes, ajouter des utilisateurs à la liste de résultats afin de rééquilibrer la charge de travail.

Avant de pouvoir utiliser les plug-in, vous devez les installer et les enregistrer. Vous pouvez enregistrer le plug-in pour permettre le post-traitement des résultats des requêtes de personnel avec l'application TaskContainer. Dans ce cas, le plug-in est disponible pour toutes les tâches.

Tâches associées

«Création de gestionnaires d'événements d'API pour Business Process Choreographer»
Un événement d'API se produit lorsqu'une méthode d'API manipule une tâche manuelle. Utilisez l'interface SPI du plug-in de gestionnaire d'événements d'API pour créer des plug-in permettant de gérer les événements de tâche envoyés par l'API ou par les événements internes ayant des événements API équivalents.

«Création des gestionnaires d'événements de notification pour Business Process Choreographer», à la page 560
Les événements de notification surviennent lors de l'escalade de tâches manuelles. Business Process Choreographer fournit des fonctionnalités permettant la gestion des escalades, telles que la création d'éléments de travail d'escalade ou l'envoi de messages électroniques. Vous pouvez créer des gestionnaires d'événements de notification pour personnaliser le mode de traitement des escalades.

«Installation des plug-in du gestionnaire d'événements d'API et du gestionnaire d'événements de notification pour les tâches manuelles», à la page 562
Pour pouvoir utiliser un plug-in de gestionnaire d'événements d'API ou de notification, vous devez l'installer pour qu'il soit accessible au conteneur de tâches manuelles.

«Enregistrement des plug-in du gestionnaire d'événements d'API et du gestionnaire d'événements de notification avec des modèles de tâche et des tâches», à la page 563
Vous pouvez enregistrer les plug-in pour les gestionnaires d'événements d'API et les gestionnaires d'événements de notification avec des tâches et des modèles de tâche à différents moments : lors de la création d'une tâche ad hoc, de la mise à jour d'une tâche existante, de la création d'un modèle de tâche ou de la définition d'un modèle de tâche.

«Utilisation d'un plug-in permettant le post-traitement des résultats d'une requête d'utilisateur», à la page 563
La résolution d'utilisateurs dans Business Process Choreographer renvoie une liste des utilisateurs auxquels un rôle spécifique est affecté, par exemple, les propriétaires potentiels d'une tâche. Vous pouvez créer un plug-in qui modifie les résultats des requêtes d'utilisateurs renvoyés par la résolution des utilisateurs. Par exemple, pour améliorer l'équilibrage de charge, vous pouvez supprimer des utilisateurs du résultat de la requête s'ils ont déjà une charge de travail élevée.

Création de gestionnaires d'événements d'API pour Business Process Choreographer

Un événement d'API se produit lorsqu'une méthode d'API manipule une tâche manuelle. Utilisez l'interface SPI du plug-in de gestionnaire d'événements d'API pour créer des plug-in permettant de gérer les événements de tâche envoyés par l'API ou par les événements internes ayant des événements API équivalents.

task_context

Exécutez les étapes suivantes pour créer un gestionnaire d'événements d'API

task_procedure

1. Ecrivez une classe qui implémente l'interface APIEventHandlerPlugin5 ou étend la classe d'implémentation APIEventHandler. Cette classe peut appeler les méthodes d'autres classes.

- Si vous utilisez l'interface APIEventHandlerPlugin5, vous devez implémenter toutes les méthodes de l'interface APIEventHandlerPlugin5 et de l'interface APIEventHandlerPlugin5.
- Si vous étendez la classe d'implémentation APIEventHandler, remplacez les méthodes selon vos besoins.

Cette classe s'exécute dans le contexte d'une application d'entreprise Java Platform, Enterprise Edition (Java EE). Assurez-vous que cette classe et ses classes helper suivent la spécification EJB.

Remarque : Pour appeler l'interface HumanTaskManagerService à partir de cette classe, n'appelez pas de méthode qui mette à jour la tâche ayant produit l'événement. Cette action peut entraîner une incohérence des données de tâche dans la base de données.

2. Assemblez la classe du plug-in et ses classes helper dans un fichier JAR.

   Pour rendre le fichier JAR disponible, vous pouvez procéder de l'une des manières suivantes :

   - En tant que fichier JAR d'utilitaire dans le fichier EAR de l'application.
   - En tant que bibliothèque partagée installée avec le fichier EAR de l'application.
   - En tant que bibliothèque partagée installée avec l'application TaskContainer. Dans ce cas, le plug-in est disponible pour toutes les tâches.

3. Créez un fichier de configuration de fournisseur de services pour le plug-in dans le répertoire `META-INF/services/` du fichier JAR.

   Le fichier de configuration fournit le mécanisme permettant d'identifier et de charger le plug-in. Ce fichier est conforme à la spécification de l'interface du fournisseur de services Java EE.

   a. Créez un fichier portant le nom`com.ibm.task.spi.`*`nom_plug-in`*`APIEventHandlerPlugin`, où *nom_plug-in* est le nom du plug-in.

      Par exemple, si votre plug-in s'appelle `Customer` et qu'il implémente l'interface com.ibm.task.spi.APIEventHandlerPlugin5, le nom du fichier de configuration est`com.ibm.task.spi.`**`Customer`**`APIEventHandlerPlugin`.

   b. La première ligne de ce fichier, qui ne doit être ni une ligne de commentaire (c'est-à-dire commençant par le signe #) ni une ligne vide, doit spécifier le nom qualifié complet de la classe de plug-in créée à l'étape 1.

      Par exemple, si la classe de votre plug-in est `MyAPIEventHandler` et se trouve dans le module `com.customer.plugins`, la première ligne du fichier de configuration doit contenir l'entrée suivante : `com.customer.plugins.MyAPIEventHandler`.

task_results

Vous avez un fichier JAR installable qui contient un plug-in gérant les événements d'API et un fichier de configuration du fournisseur de services pouvant être utilisé pour charger le plug-in.

Remarques : Vous ne disposez que d'une propriété eventHandlerName pour enregistrer à la fois les gestionnaires d'événements d'API et les gestionnaires d'événements de notification. Pour utiliser à la fois un gestionnaire d'événement d'API If et un gestionnaire d'événement de notification, il est nécessaire que les implémentations des plug-ins portent le même nom (`Customer` comme nom de gestionnaire d'événement pour l'implémentation de SPI, par exemple).

Vous pouvez implémenter les deux plug-ins à l'aide d'une seule classe ou de classes distinctes. Dans les deux cas, vous devez créer deux fichiers dans le répertoire `META-INF/services/` de votre fichier JAR (par exemple, com.ibm.task.spi.CustomerNotificationEventHandlerPlugin et com.ibm.task.spi.CustomerAPIEventHandlerPlugin).

Regroupez l'implémentation du plug-in et les classes helper dans un seul fichier JAR.

Pour rendre effective une modification de l'implémentation, remplacez le fichier JAR contenu dans la bibliothèque partagée, déployez à nouveau le fichier EAR associé et redémarrez le serveur.

task_postreq

Vous devez maintenant installer et enregistrer le plug-in afin de le rendre disponible pour le conteneur de tâches manuelles lors de l'exécution. Vous pouvez enregistrer des gestionnaires d'événements liés à l'API avec une instance de tâche, un modèle de tâche ou un composant d'application.

Concepts associés

«Gestionnaires d'événements d'API»
Les événements d'API surviennent lorsqu'une tâche manuelle est modifiée ou change d'état. Pour permettre le traitement de ces événements d'API, le gestionnaire d'événements est appelé directement avant la modification de la tâche (méthode pré-événement) et juste après le renvoi de l'appel API (méthode post-événement).

Gestionnaires d'événements d'API

Les événements d'API surviennent lorsqu'une tâche manuelle est modifiée ou change d'état. Pour permettre le traitement de ces événements d'API, le gestionnaire d'événements est appelé directement avant la modification de la tâche (méthode pré-événement) et juste après le renvoi de l'appel API (méthode post-événement).

Si la méthode pré-événement génère une exception ApplicationVetoException, l'action de l'API n'est pas exécutée, l'exception est renvoyée à l'appelant de l'API et la transaction associée à l'événement est annulée. Si la méthode pré-événement a été déclenchée par un événement interne et qu'une exception ApplicationVetoException est générée, l'événement interne (par exemple une réclamation automatique) n'est pas exécuté mais une exception est renvoyée à l'application client. Dans ce cas, un message d'information est enregistré dans le fichier SystemOut.log. Si la méthode d'API génère une exception au cours du traitement, celle-ci est interceptée et transmise à la méthode post-événement. L'exception est de nouveau transmise à l'appelant lorsque la méthode post-événement est renvoyée.

Les règles suivantes s'appliquent aux méthodes pré-événement :

- Les méthodes pré-événement reçoivent les paramètres de la méthode d'API ou de l'événement interne associé(e).
- Les méthodes pré-événement peuvent générer une exception ApplicationVetoException pour empêcher la poursuite du traitement.

Les règles suivantes s'appliquent aux méthodes post-événement :

- Les méthodes post-événement reçoivent les paramètres fournis à l'appel d'API, puis renvoient les valeurs. Si une exception est émise par l'implémentation d'une méthode d'API, la méthode post-événement reçoit également l'exception.
- Les méthodes post-événement ne modifient pas les valeurs renvoyées.
- Les méthodes post-événement ne peuvent pas générer d'exceptions. Les exceptions d'exécution sont consignées et empêche le traitement de se poursuivre.

Pour implémenter les gestionnaires d'événements d'API, vous pouvez au choix faire appel à l'interface APIEventHandlerPlugin3, qui étend l'interface APIEventHandlerPlugin, ou bien étendre la classe d'implémentation SPI par défaut com.ibm.task.spi.APIEventHandler. Si votre gestionnaire d'événements hérite de la classe d'implémentation par défaut, il implémente toujours la version la plus récente de l'interface SPI. Si vous effectuez une mise à niveau vers une version plus récente de Business Process Choreographer, quelques modifications doivent être apportées si vous souhaitez utiliser de nouvelles méthodes d'interface SPI.

Si un gestionnaire d'événements de notification et un gestionnaire d'événements d'API sont présents simultanément, ils doivent tous deux porter le même nom, car il n'est possible de nommer qu'un seul gestionnaire.

Création des gestionnaires d'événements de notification pour Business Process Choreographer

Les événements de notification surviennent lors de l'escalade de tâches manuelles. Business Process Choreographer fournit des fonctionnalités permettant la gestion des escalades, telles que la création d'éléments de travail d'escalade ou l'envoi de messages électroniques. Vous pouvez créer des gestionnaires d'événements de notification pour personnaliser le mode de traitement des escalades.

task_context

Pour implémenter les gestionnaires d'événements de notification, vous pouvez soit faire appel à l'interface NotificationEventHandlerPlugin, soit dériver la classe d'implémentation SPI par défaut com.ibm.task.spi.NotificationEventHandler.

Suivez la procédure ci-après pour créer un gestionnaire d'événements de notification.

task_procedure

1. Générez une classe qui implémente l'interface NotificationEventHandlerPlugin ou étend la classe d'implémentation NotificationEventHandler. Cette classe permet d'appeler les méthodes des autres classes.

   Si vous utilisez l'interface NotificationEventHandlerPlugin, vous devez implémenter toutes les méthodes de cette interface. Si vous étendez la classe d'implémentation SPI, remplacez les méthodes selon vos besoins.

   Cette classe s'exécute dans le contexte d'une application d'entreprise Java Platform, Enterprise Edition (Java EE). Assurez-vous que cette classe et ses classes helper suivent les la spécification EJB.

   Le plug-in est appelé avec les droits d'accès associés au rôle EscalationUser. Ce rôle est défini lorsque le conteneur des tâches manuelles est configuré.

Remarque : Pour appeler l'interface HumanTaskManagerService à partir de cette classe, n'appelez pas de méthode qui mette à jour la tâche ayant produit l'événement. Cette action peut entraîner une incohérence des données de tâche dans la base de données.

2. Assemblez la classe du plug-in et ses classes helper dans un fichier JAR.

   Pour rendre le fichier JAR disponible, vous pouvez procéder de l'une des manières suivantes :

   - En tant que fichier JAR d'utilitaire dans le fichier EAR de l'application.
   - En tant que bibliothèque partagée installée avec le fichier EAR de l'application.
   - En tant que bibliothèque partagée installée avec l'application TaskContainer. Dans ce cas, le plug-in est disponible pour toutes les tâches.

3. Assemblez la classe du plug-in et ses classes helper dans un fichier JAR.

   Si les classes helper sont utilisées par plusieurs applications Java EE, vous pouvez les regrouper dans un fichier JAR distinct que vous enregistrez sous forme de bibliothèque partagée.

4. Créez un fichier de configuration de fournisseur de services pour le plug-in dans le répertoire `META-INF/services/` de votre fichier JAR.

   Le fichier de configuration fournit le mécanisme d'identification et de chargement du plug-in. Ce fichier est conforme à la spécification de l'interface du fournisseur de services Java EE.

   a. Créez un fichier nommé `com.ibm.task.spi.`*`nom_plug-in`*`NotificationEventHandlerPlugin`, ou *nom_plug-in* est le nom du plug-in.

      Si, par exemple, votre plug-in est nommé `HelpDeskRequest` (nom du gestionnaire d'événements) et qu'il implémente l'interface com.ibm.task.spi.NotificationEventHandlerPlugin, le fichier de configuration porte le nom `com.ibm.task.spi.`**`HelpDeskRequest`**`NotificationEventHandlerPlugin`.

   b. La première ligne de ce fichier, qui ne doit être ni une ligne de commentaire (c'est-à-dire commençant par le signe #) ni une ligne vide, doit spécifier le nom qualifié complet de la classe de plug-in créée à l'étape 1.

      Si par exemple la classe de plug-in porte le nom `MyEventHandler` et est incluse dans le package `com.customer.plugins`, la première ligne du fichier de configuration doit contenir l'entrée suivante : `com.customer.plugins.MyEventHandler`.

task_results

Vous disposez d'un fichier JAR installable contenant un plug-in qui gère les événements de notification et d'un fichier de configuration de fournisseur de services pouvant servir à charger le plug-in. Vous pouvez enregistrer des gestionnaires d'événements liés à l'API avec une instance de tâche, un modèle de tâche ou un composant d'application.

Remarques : Vous ne disposez que d'une propriété eventHandlerName pour enregistrer à la fois les gestionnaires d'événements d'API et les gestionnaires d'événements de notification. Pour utiliser à la fois un gestionnaire d'événement d'API If et un gestionnaire d'événement de notification, il est nécessaire que les implémentations des plug-ins portent le même nom (`Customer` comme nom de gestionnaire d'événement pour l'implémentation de SPI, par exemple).

Vous pouvez implémenter les deux plug-ins à l'aide d'une seule classe ou de classes distinctes. Dans les deux cas, vous devez créer deux fichiers dans le

répertoire `META-INF/services/` de votre fichier JAR (par exemple, com.ibm.task.spi.CustomerNotificationEventHandlerPlugin et com.ibm.task.spi.CustomerAPIEventHandlerPlugin).

Regroupez l'implémentation du plug-in et les classes helper dans un seul fichier JAR.

Pour rendre effective une modification de l'implémentation, remplacez le fichier JAR contenu dans la bibliothèque partagée, déployez à nouveau le fichier EAR associé et redémarrez le serveur.

task_postreq

Vous devez maintenant installer et enregistrer le plug-in afin de le rendre disponible pour le conteneur de tâches manuelles lors de l'exécution. Vous pouvez enregistrer des gestionnaires d'événements de notification avec une instance de tâche, un modèle de tâche ou un composant d'application.

Installation des plug-in du gestionnaire d'événements d'API et du gestionnaire d'événements de notification pour les tâches manuelles

Pour pouvoir utiliser un plug-in de gestionnaire d'événements d'API ou de notification, vous devez l'installer pour qu'il soit accessible au conteneur de tâches manuelles.

task_context

Le mode d'installation du plug-in varie suivant qu'il doit être utilisé par une seule application Java Platform, Enterprise Edition (Java EE) ou par plusieurs applications.

Procédez de l'une des manières suivantes pour installer un plug-in.

task_procedure

- Installez plug-in pour qu'il soit utilisé par une seule application Java EE.

  Ajoutez le fichier JAR du plug-in au fichier JAR de l'application. Dans l'éditeur du descripteur de déploiement de WebSphere Integration Developer, installez le fichier JAR de votre plug-in en tant que fichier JAR d'utilitaire de projet pour l'application Java EE du module EJB (JavaBeans) principal.
- Installez un plug-in pour qu'il soit utilisé par plusieurs applications Java EE.

  Placez le fichier JAR dans une bibliothèque partagée de WebSphere Application Server et associez la bibliothèque aux applications devant accéder au plug-in. Pour rendre le fichier JAR accessible dans un environnement de déploiement réseau, distribuez manuellement le fichier JAR sur chaque noeud hébergeant un serveur ou un membre de cluster sur lequel l'une de vos applications est déployée. Vous pouvez utiliser la portée de la cible de déploiement de vos applications, c'est-à-dire le serveur ou le cluster sur lequel les applications sont déployées, ou bien la portée de cellule. Souvenez-vous que les classes des plug-in sont alors visibles dans toute la portée de déploiement sélectionnée.

task_postreq

Vous pouvez, maintenant, enregistrer le plug-in.

Enregistrement des plug-in du gestionnaire d'événements d'API et du gestionnaire d'événements de notification avec des modèles de tâche et des tâches

Vous pouvez enregistrer les plug-in pour les gestionnaires d'événements d'API et les gestionnaires d'événements de notification avec des tâches et des modèles de tâche à différents moments : lors de la création d'une tâche ad hoc, de la mise à jour d'une tâche existante, de la création d'un modèle de tâche ou de la définition d'un modèle de tâche.

task_context

Vous pouvez enregistrer des plug-in pour les gestionnaires d'événements d'API et les gestionnaires d'événements de notification avec des tâches à différents niveaux :

Modèle de tâche
Toutes les tâches créées à l'aide du modèle utilisent les mêmes gestionnaires

Modèle de tâche ad hoc
Les tâches créées à l'aide du modèle utilisent les mêmes gestionnaires

Tâche ad hoc
La tâche créée utilise les gestionnaires spécifiés

Tâche existante
La tâche utilise les gestionnaires spécifiés

Vous pouvez enregistrer un plug-in en suivant l'une des procédures suivantes.

task_procedure

- Pour les modèles de tâches modélisés dans WebSphere Integration Developer, spécifiez le plug-in dans le modèle de tâche.
- Pour les tâches ad hoc ou les modèles de tâches ad hoc, indiquez le plug-in au moment de la création de la tâche ou du modèle de tâche.

  Utilisez la méthode setEventHandlerName de la classe TTask pour enregistrer le nom du gestionnaire d'événements.
- Modifiez le gestionnaire d'événements pour une instance de tâche lors de l'exécution.

  La méthode update(Task task) vous permet d'utiliser un autre gestionnaire d'événements pour une instance de tâche lors de l'exécution. L'appelant doit disposer de droit d'accès administrateur pour mettre à jour cette propriété.

Utilisation d'un plug-in permettant le post-traitement des résultats d'une requête d'utilisateur

La résolution d'utilisateurs dans Business Process Choreographer renvoie une liste des utilisateurs auxquels un rôle spécifique est affecté, par exemple, les propriétaires potentiels d'une tâche. Vous pouvez créer un plug-in qui modifie les résultats des requêtes d'utilisateurs renvoyés par la résolution des utilisateurs. Par exemple, pour améliorer l'équilibrage de charge, vous pouvez supprimer des utilisateurs du résultat de la requête s'ils ont déjà une charge de travail élevée.

task_context

Pour modifier les résultats renvoyés par l'affectation et la substitution des utilisateurs, vous devez écrire une classe qui implémente l'interface du plug-in, assembler un fichier JAR file pour le plug-in, puis l'installer et l'activer.

Exécutez les étapes suivantes pour créer un plug-in permettant le post-traitement des résultats d'une requête d'utilisateur.

task_procedure

1. Implémentez votre plug-in de post-traitement des résultats de requête utilisateur. Ecrivez une classe qui implémente l'interface StaffQueryResultPostProcessorPlugin ou StaffQueryResultPostProcessorPlugin2.
2. Créez un fichier JAR installable.
   a. Assemblez votre classe de plug-in et ses classes helper dans un fichier JAR.
   b. Créez un fichier de configuration de fournisseur de services pour le plug-in dans le répertoire `META-INF/services/` du fichier JAR. Le fichier de configuration fournit le mécanisme permettant d'identifier et de charger le plug-in. Ce fichier doit être conforme à la spécification de l'interface du fournisseur de services Java EE.
      1) Dans un éditeur de texte, créez un fichier de configuration de fournisseur de services intitulé `com.ibm.task.spi.`*`nom_plug-in`*`StaffQueryResultPostProcessorPlugin`, *nom_plug-in* représentant le nom du plug-in. Le nom du fichier de configuration ne dépend pas du nom de l'interface que vous avez implémentée. Par exemple, si votre plug-in s'appelle `MyHandler` et qu'il implémente l'interface com.ibm.task.spi.StaffQueryResultPostProcessorPlugin2, le nom du fichier de configuration sera `com.ibm.task.spi.`**`MyHandler`**`StaffQueryResultPostProcessorPlugin`.
      2) Dans la première ligne du fichier qui ne correspond pas à une ligne de commentaire (ligne commençant par le signe dièse (#)) ou à une ligne vide, spécifiez le nom complet de la classe de plug-in que vous avez créée à l'étape 1. Par exemple, si la classe de votre plug-in est `StaffPostProcessor` et se trouve dans le module `com.customer.plugins`, la première ligne du fichier de configuration doit contenir l'entrée suivante : `com.customer.plugins.StaffPostProcessor`.

   Vous avez un fichier JAR installable qui contient un plug-in assurant le post-traitement des résultats de requête d'utilisateur et un fichier de configuration du fournisseur de services pouvant être utilisé pour charger le plug-in.
3. Installez le fichier JAR dans une bibliothèque partagée du serveur d'applications et associez-le à l'application Human Task Manager.
   a. Définissez une bibliothèque partagée WebSphere Application Server pour le plug-in dans la portée du serveur ou du cluster sur lequel Business Process Choreographer est configuré.
   b. Associez la bibliothèque partagée à l'application `TaskContainer`.
   c. Rendez le fichier JAR de plug-in disponible pour chaque installation de WebSphere Process Server concernée qui héberge un serveur ou un membre de cluster.
4. Configurez Human Task Manager pour qu'il utilise le plug-in.
   a. Dans la console d'administration, accédez à la page Propriétés personnalisées de Human Task Manager.

Cliquez sur soit **Serveurs** → **Clusters** → **Clusters de serveurs d'applications WebSphere** → *nom_cluster* ou **Serveurs** → **Types de serveur** → **WebSphere application servers** → *nom_serveur*, puis dans l'onglet **Configuration**, sous la section **Business Integration**, développez **Business Process Choreographer** et cliquez sur **Human Task Manager**. Dans **Propriétés supplémentaires**, cliquez sur **Propriétés personnalisées**.

b. Ajoutez une propriété personnalisée nommée **Staff.PostProcessorPlugin** et ainsi que la valeur du nom que vous avez donné à votre plug-in (par exemple, `MyHandler`).

Le plug-in est désormais disponible pour effectuer le post-traitement des résultats de requête du personnel.

5. Redémarrez le serveur pour activer le plug-in. Le plug-in de post-traitement est appelé une fois que les opérations d'affectation et de substitution des utilisateurs sont terminées.

Remarque : Si vous modifiez le plug-in, vous devez remplacer le fichier JAR dans la bibliothèque partagée, puis redémarrer le serveur.

Partie 2. Déploiement des applications

Chapitre 5. Présentation de la préparation et de l'installation de modules

L'installation de modules (également appelée déploiement) active ces modules soit dans un environnement de test, soit dans un environnement de production. Cette présentation décrit brièvement les environnements de test et de production, ainsi que certaines étapes de l'installation de modules.

Remarque : Le processus d'installation d'applications dans un environnement de production est similaire au processus décrit dans la rubrique «Développement et déploiement d'applications» du centre de documentation de WebSphere Application Server Network Deployment for z/OS. Si vous ne connaissez pas ces rubriques, reportez-vous y en premier.

Avant d'installer un module dans un environnement de production, vérifiez à chaque fois les modifications dans un environnement de test. Pour installer des modules dans un environnement de test, utilisez WebSphere Integration Developer (voir le centre de documentation WebSphere Integration Developer pour plus d'informations). Pour installer des modules dans un environnement de production, utilisez WebSphere Process Server.

Cette rubrique décrit les concepts et les tâches nécessaires à la préparation et à l'installation de modules dans un environnement de production. Les autres rubriques décrivent les fichiers contenant les objets que votre module utilise et vous aide à déplacer ce module de l'environnement de test vers l'environnement de production. Il est important de comprendre ces fichiers et leur contenu pour être sûr d'avoir installé vos modules correctement.

Concepts associés

Présentation des bibliothèques et des fichiers JAR
Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques, qui correspondent à des projets spéciaux de WebSphere Integration Developer utilisés pour le stockage des ressources partagées. Lors de la phase de déploiement, les bibliothèques de WebSphere Integration Developer sont transformées en fichiers JAR d'utilitaire et intégrées aux applications à exécuter.

Présentation du fichier EAR
Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Tâches associées

Préparation au déploiement sur un serveur
Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Information associée

Remarques concernant l'installation d'applications de service sur des clusters
L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Présentation des bibliothèques et des fichiers JAR

Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques, qui correspondent à des projets spéciaux de WebSphere Integration Developer utilisés pour le stockage des ressources partagées. Lors de la phase de déploiement, les bibliothèques de WebSphere Integration Developer sont transformées en fichiers JAR d'utilitaire et intégrées aux applications à exécuter.

Lors du développement d'un module, il est possible d'identifier certaines ressources ou composants qui peuvent être utilisés par d'autres modules. Ces artefacts peuvent être partagés à l'aide d'une bibliothèque.

Bibliothèque

Une bibliothèque est un projet spécial de WebSphere Integration Developer utilisé pour le développement, la gestion des versions et l'organisation des ressources partagées, telles que les ressources généralement partagées entre les modules. Seul un sous-ensemble des types d'artefact peut être créé et stocké dans une bibliothèque, et notamment :

- des interfaces ou des descripteurs de services Web (fichiers ayant une extension .wsdl) ;
- des définitions de schéma XML d'objets métier (fichiers ayant une extension .xsd) ;
- des mappes d'objets métier (fichiers ayant une extension .map) ;
- des définitions de relations et de rôles (fichiers ayant une extension .rel et .rol).

Lors de la phase de déploiement, ces bibliothèques de WebSphere Integration Developer sont transformées en fichiers JAR d'utilitaire dans les applications à exécuter.

Lorsqu'un module doit utiliser un artefact, le serveur recherche cet artefact à partir du chemin d'accès aux classes EAR et le charge, s'il n'est pas déjà chargé dans la

mémoire. La figure 78 illustre les composants et les bibliothèques d'une application.

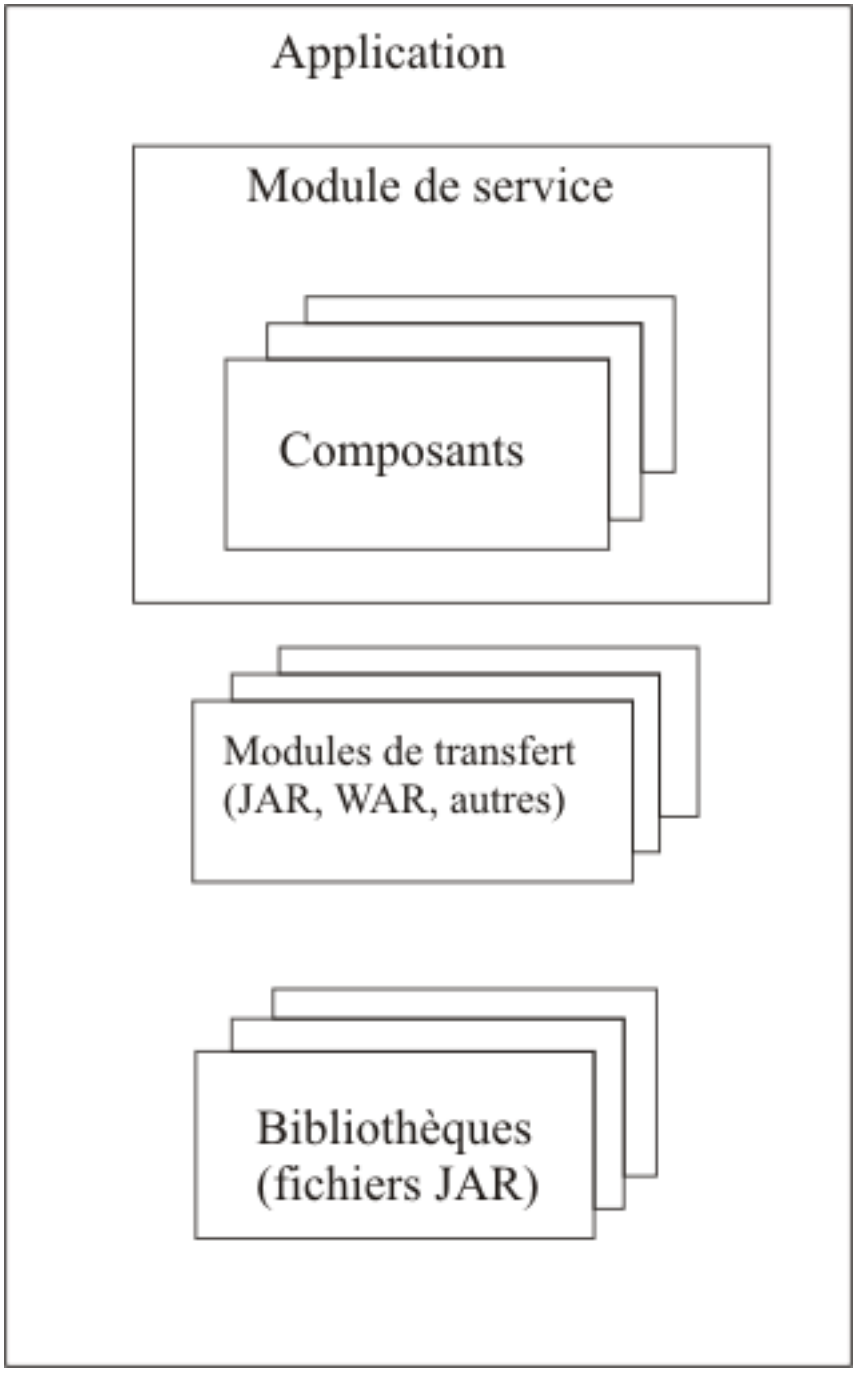


Figure 78. Relations entre module, composant et bibliothèque

Fichiers JAR, RAR et WAR

Un certain nombre de fichiers peuvent contenir des composants d'un module. Ces fichiers sont décrits en détails dans la spécification Java Platform, Enterprise Edition (J2EE). Une description détaillée des fichiers JAR est disponible dans la spécification JAR.

Dans WebSphere Process Server, un fichier JAR contient également une application qui est la version assemblée du module comprenant toutes les références de prise en charge et les interfaces vers tous les autres composants de service utilisés par le module. Pour installer l'application complète, vous avez besoin de ce fichier JAR et des fichiers JAR dépendants, des archives de service Web (WAR), de l'archive des ressources (RAR), des fichiers JAR des bibliothèques de transfert (Enterprise Java Beans - EJB) et de toutes autres archives. Vous créez ensuite un fichier EAR installable à l'aide de la commande serviceDeploy.

Conventions de dénomination pour les modules de transfert

Dans la bibliothèque, des conventions de dénomination s'appliquent aux noms des modules de transfert. Ces noms sont uniques pour un module spécifique. Nommez les autres modules requis pour déployer l'application en veillant à éviter tout conflit avec les noms des modules de transfert. Pour un module nommé *myService*, les noms de modules de transfert sont les suivants :

- *myService*App
- *myService*Web

Remarque : Les modules de transfert *myService*EJB et *myService*EJBClient ne sont plus créés par serviceDeploy. Toutefois, ces noms de fichier ne doivent pas être utilisés car ils peuvent toujours être supprimés par la commande serviceDeploy.

Remarques concernant l'utilisation de bibliothèques

L'utilisation de bibliothèques assure la cohérence des objets métier et celle du traitement entre les différents modules étant donné que chaque module appelant dispose de sa propre copie d'un composant spécifique. Pour empêcher les incohérences et les erreurs, il est important de veiller à ce que les modifications apportées aux composants et aux objets métiers utilisés par les modules appelants soient coordonnées avec l'ensemble des modules appelants. Pour mettre les modules appelants à jour, procédez comme suit :

1. copiez le module et la copie la plus récente des bibliothèques sur le serveur de production ;
2. recréez le fichier EAR installable à l'aide de la commande serviceDeploy ;
3. arrêtez l'application en cours d'exécution qui contient le module appelant et réinstallez-la ;
4. redémarrez l'application qui contient le module appelant.

Concepts associés

Présentation du fichier EAR
Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Tâches associées

Préparation au déploiement sur un serveur
Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Information associée

Remarques concernant l'installation d'applications de service sur des clusters
L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Présentation du fichier EAR

Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Un fichier d'archive d'entreprise (EAR) est un fichier compressé qui contient les bibliothèques, les beans enterprise et les fichiers JAR nécessaires au déploiement de l'application.

Les fichiers JAR sont créés lors de l'exportation des modules d'application à partir de WebSphere Integration Developer. Ce fichier JAR et toutes autres bibliothèques d'artefacts ou objets sont utilisés en tant qu'entrées dans le processus d'installation. La commande serviceDeploy crée un fichier EAR à partir des fichiers d'entrée contenant les descriptions des composants et le code Java qui forment l'application.

Concepts associés

Présentation des bibliothèques et des fichiers JAR
Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques, qui correspondent à des projets spéciaux de WebSphere Integration Developer utilisés pour le stockage des ressources partagées. Lors de la phase de déploiement, les bibliothèques de WebSphere Integration Developer sont transformées en fichiers JAR d'utilitaire et intégrées aux applications à exécuter.

Tâches associées

Préparation au déploiement sur un serveur
Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Information associée

Remarques concernant l'installation d'applications de service sur des clusters
L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Préparation au déploiement sur un serveur

Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

task_prereq

Avant de commencer, vous devez avoir développé et testé vos modules sur un serveur de test et résolu les incidents et les problèmes liés aux performances.

Important : Pour éviter de remplacer une application ou un module s'exécutant déjà dans un environnement de déploiement, assurez-vous que le nom du module ou de l'application est différent de celui déjà installé.

task_context

Cette tâche vérifie que toutes les pièces nécessaires d'une application sont disponibles et rassemblées dans les bons fichiers pour être amenées vers le serveur de production.

Remarque : Vous pouvez également exporter un fichier d'archive d'entreprise (EAR) à partir de WebSphere Integration Developer et installer ce fichier directement dans WebSphere Process Server.

Important : Si les services internes d'un composant utilisent une base de données, installez l'application sur un serveur connecté directement à une base de données.

task_procedure

1. Localisez le dossier contenant les composants du module que vous souhaitez déployer.

   Le dossier contenant les composants doit porter le nom *module-nom*et contenir un fichier nommé *module*.module correspondant au module de base.

2. Vérifiez que tous les composants contenus dans le module se trouvent dans les sous-dossiers de composant sous le dossier du module.

   Pour faciliter l'utilisation, nommez le sous-dossier de la façon suivante *module*/*composant*.

3. Vérifiez que tous les fichiers du composants figurent dans le sous-dossier de composant approprié et que leur nom ressemble à *nom-fichier-composant*.composant.

   Les fichiers de composants contiennent les définitions de chaque composant individuel à l'intérieur du module.

4. Vérifiez que tous les autres composants et artefacts se trouvent bien dans les sous-dossiers de composants qui exigent leur présence.

   Lors de cette étape, vous allez vérifier que toutes les références à des outils nécessaires à un composant sont disponibles. Les noms de composants ne doivent pas entrer en conflit avec les noms que la commande serviceDeploy utilise pour hiérarchiser les modules. Voir convention de dénomination des modules de transfert.

5. Vérifiez que le fichier de références, *module*.references, existe bien dans le dossier module de l'étape 1, à la page 574.

   Le fichier de références définit les références et les interfaces à l'intérieur du module.

6. Vérifiez que le fichier câblage, *module*.wires, existe bien dans le dossier composant.

   Le fichier câblage complète les connexions entre les références et les interfaces du module.

7. Vérifiez que le fichier manifeste, *module*.manifest, existe bien dans le dossier composant.

   Le manifeste répertorie le module et tous les composants du module. Il contient également une instruction de chemin de classes afin de permettre à la commande serviceDeploy de localiser tout autre module nécessaire au module.

8. Créez un fichier compressé ou un fichier JAR du module représentant l'entrée de la commande serviceDeploy que vous utiliserez afin de préparer l'installation du module vers le serveur de production.

Exemple de structure de dossiers pour le module MyValue avant le déploiement

Ce qui suit illustre la structure de répertoire du module MyValueModule comprenant les composants MyValue, CustomerInfo et StockQuote.

```
MyValueModule
   MyValueModule.manifest
   MyValueModule.references
   MyValueModule.wiring
   MyValueClient.jsp
process/myvalue
   MyValue.component
   MyValue.java
   MyValueImpl.java
service/customerinfo
   CustomerInfo.component
   CustomerInfo.java
   Customer.java
   CustomerInfoImpl.java
service/stockquote
   StockQuote.component
```

```
StockQuote.java
StockQuoteAsynch.java
StockQuoteCallback.java
StockQuoteImpl.java
```

task_postreq

Installez le module sur les systèmes de production comme décrit à la rubrique Installation d'un module sur un serveur de production.

Concepts associés

Présentation des bibliothèques et des fichiers JAR
Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques, qui correspondent à des projets spéciaux de WebSphere Integration Developer utilisés pour le stockage des ressources partagées. Lors de la phase de déploiement, les bibliothèques de WebSphere Integration Developer sont transformées en fichiers JAR d'utilitaire et intégrées aux applications à exécuter.

Présentation du fichier EAR
Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Information associée

Remarques concernant l'installation d'applications de service sur des clusters
L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Remarques concernant l'installation d'applications de service sur des clusters

L'installation d'une application de service sur un cluster implique d'autres exigences. Il est important de les garder à l'esprit lors de l'installation d'applications de service sur un cluster.

Les clusters apportent de nombreux avantages à votre environnement de traitement grâce aux économies d'échelle, ce qui permet d'équilibrer la charge des requêtes entre serveurs et fournit un niveau de disponibilité pour les clients des applications. Avant d'installer une application contenant des services sur un cluster, tenez compte des points suivants :

- Les utilisateurs de l'applications ont-ils besoin de la puissance et de la disponibilité de traitement des clusters ?

  Si c'est le cas, la mise en cluster est la solution adéquate. La mise en cluster augmente la disponibilité et la capacité de vos applications.
- Le cluster est-il préparé correctement pour les applications de service ?

  Vous devez configurer le cluster correctement avant d'installer et de démarrer la première application contenant un service. Le cluster doit être configuré correctement pour que les requêtes soient traitées correctement.
- Un cluster de secours est-il installé ?

  Vous devez installer l'application sur le cluster de secours également.

Concepts associés

Présentation des bibliothèques et des fichiers JAR
Les modules utilisent souvent des artefacts qui se trouvent dans des bibliothèques, qui correspondent à des projets spéciaux de WebSphere Integration Developer utilisés pour le stockage des ressources partagées. Lors de la phase de déploiement, les bibliothèques de WebSphere Integration Developer sont transformées en fichiers JAR d'utilitaire et intégrées aux applications à exécuter.

Présentation du fichier EAR
Un fichier EAR est un élément critique du déploiement d'une application de service sur un serveur de production.

Tâches associées

Préparation au déploiement sur un serveur
Après avoir développé et testé un module, vous devez l'exporter d'un système de test vers un environnement de production en vue de son déploiement. Pour installer une application, vous devez également déterminer les chemins requis lors de l'exportation du module et les bibliothèques requises par celui-ci.

Chapitre 6. Installation des applications de tâche manuelle et de processus métier

Vous pouvez distribuer les modules SCA (Service Component Architecture) contenant des processus métier ou des tâches manuelles, ou les deux, sur des cibles de déploiement. Une cible de déploiement peut être un serveur ou un cluster.

task_prereq

Vérifiez que Business Flow Manager et Human Task Manager sont installés et configurés pour chaque serveur d'applications ou cluster sur lequel vous souhaitez installer l'application.

task_context

Vous pouvez installer des applications de processus métier et de tâche à partir de la console d'administration ou de la ligne de commande, ou en exécutant un script d'administration.

task_results

Après l'installation d'une application de processus métier ou de tâche manuelle, tous les modèles de processus métier et de tâche manuelle passent à l'état "Démarré". Vous pouvez créer des instances de processus et de tâche à partir de ces modèles.

task_postreq

Pour pouvoir créer des instances de processus ou de tâche, vous devez démarrer l'application.

Concepts associés

«Installation d'applications de processus métier et de tâches manuelles dans un environnement de déploiement réseau»
Lorsque des modèles de processus ou de tâches manuelles sont installés dans un environnement de déploiement réseau, les actions suivantes sont automatiquement exécutées par le programme d'installation des applications.

«Déploiement des processus métier et des tâches manuelles», à la page 581
Utilisez WebSphere Integration Developer ou serviceDeploy pour mettre en forme les composants de processus ou de tâches dans un fichier EAR. Chaque nouvelle version d'un modèle devant être déployé doit être mise en forme dans une nouvelle application d'entreprise.

Tâches associées

«Installation d'applications de processus métier et de tâche manuelle en mode interactif», à la page 581
Vous pouvez installer une application en mode interactif lors son exécution à l'aide de l'outil wsadmin et du script installInteractive. Vous pouvez utiliser le script pour modifier les paramètres qui ne sont pas modifiables si vous utilisez la console d'administration pour installer l'application.

«Désinstallation d'applications de processus métier et de tâche manuelle à l'aide de la console d'administration», à la page 584
Vous pouvez utiliser la console d'administration pour désinstaller des applications contenant des processus métier ou des tâches manuelles.

«Désinstallation d'applications de processus métier et de tâches manuelles à l'aide d'une commande d'administration», à la page 585
L'utilisation du script bpcTemplates.jacl est une alternative à l'emploi de la console d'administration pour désinstaller des applications qui contiennent des processus métier ou des tâches manuelles.

Installation d'applications de processus métier et de tâches manuelles dans un environnement de déploiement réseau

Lorsque des modèles de processus ou de tâches manuelles sont installés dans un environnement de déploiement réseau, les actions suivantes sont automatiquement exécutées par le programme d'installation des applications.

L'application est installée par étapes. Chaque étape doit être exécutée avec succès pour que la suivante puisse débuter.

1. L'installation d'application démarre sur le gestionnaire de déploiement.

   Au cours de cette étape, les modèles de processus métier et de tâche manuelle sont configurés dans le référentiel de configuration WebSphere. L'application est également validée. Si des erreurs se produisent, elles sont consignées dans les fichiers System.out et System.err, ou en tant qu'entrées FFDC dans le gestionnaire de déploiement.
2. L'installation de l'application se poursuit sur l'agent de noeud.

   Au cours de cette étape, l'installation de l'application sur une instance de serveur d'applications est déclenchée. Cette instance de serveur d'applications est soit la cible de déploiement, soit fait partie de celle-ci. Si la cible de déploiement est un cluster comprenant plusieurs membres, l'instance du serveur est choisie arbitrairement parmi les membres de ce cluster. Si des erreurs se produisent au cours de cette étape, elles sont consignées dans les fichiers SystemOut.log et SystemErr.log, ou en tant qu'entrées FFDC sur l'agent de noeud.
3. L'application s'exécute sur l'instance de serveur.

Au cours de cette étape, les modèles de processus métier et de tâche manuelle sont déployés dans la base de données de Business Process Choreographer sur la cible de déploiement. Si des erreurs se produisent, elles sont consignées dans les fichiers System.out et SystemErr.log ou en tant qu'entrées FFDC sur l'instance de serveur.

Déploiement des processus métier et des tâches manuelles

Utilisez WebSphere Integration Developer ou serviceDeploy pour mettre en forme les composants de processus ou de tâches dans un fichier EAR. Chaque nouvelle version d'un modèle devant être déployé doit être mise en forme dans une nouvelle application d'entreprise.

Lorsque vous installez une application d'entreprise qui contient des processus métier ou des tâches manuelles, ces derniers sont stockés dans des modèles de processus métier ou des modèles de tâches manuelles, au sein de la base de données du Business Process Choreographer. Les modèles nouvellement installés sont, par défaut, à l'état démarré. Toutefois, l'application d'entreprise nouvellement installée se trouve à l'état arrêté. Chaque application d'entreprise installée peut être démarrée et arrêtée individuellement.

Vous pouvez déployer de nombreuses versions différentes d'un modèle de processus ou de tâche, chacune dans un application d'entreprise différente. Les versions sont différenciées en fonction de leur date de début de validité. Lorsque vous installez une nouvelle application d'entreprise, la version du modèle qui est installée est déterminée comme suit :

- Si le nom du modèle et l'espace de nom cible n'existent pas, un nouveau modèle est installé.
- Si le nom du modèle et l'espace de nom cible sont identiques à ceux du modèle existant, mais que la date de début de validité est différente, une nouvelle version du modèle existant est installée.

Remarque : Le nom du modèle est dérivé du nom du composant et non du processus métier ou de la tâche manuelle.

Si vous n'indiquez pas de date de début de validité, la date est déterminée de la façon suivante :

- Si vous utilisez WebSphere Integration Developer, la date de début de validité correspond à la date de modélisation de la tâche manuelle ou du processus métier.
- Si vous utilisez le déploiement de service, la date de début de validité correspond à la date d'exécution de la commande serviceDeploy. Seules les tâches collaboratives affichent la date d'installation de l'application comme date de début de validité.

Installation d'applications de processus métier et de tâche manuelle en mode interactif

Vous pouvez installer une application en mode interactif lors son exécution à l'aide de l'outil wsadmin et du script installInteractive. Vous pouvez utiliser le script pour modifier les paramètres qui ne sont pas modifiables si vous utilisez la console d'administration pour installer l'application.

task_context

Procédez comme suit pour installer des applications de processus métier en mode interactif.

task_procedure

1. Démarrez l'outil wsadmin.

   Dans le répertoire *racine_profil*/bin, entrez wsadmin.
2. Installez l'application.

   Dans l'invite de ligne de commande, entrez la commande suivante :

   ```
   $AdminApp installInteractive application.ear
   ```

   où *application.ear* désigne le nom qualifié du fichier EAR (Enterprise Archive) contenant votre application de processus. Une série de tâches vous permet de modifier les valeurs définies pour l'application.
3. Sauvegardez les modifications apportées à la configuration.

   Dans l'invite de ligne de commande, entrez la commande suivante :

   ```
   $AdminConfig save
   ```

   Vous devez sauvegarder vos modifications afin de transférer les mises à jour au référentiel de configuration maître. Si un processus de scriptage se termine et que vous n'avez pas sauvegardé vos modifications, celles-ci sont supprimées.

Tâches associées

«Configuration de la source de données d'une application de processus et des paramètres de référence d'ensemble»
Il peut être nécessaire de configurer les applications de processus exécutant des instructions SQL pour une infrastructure de base de données spécifique. Ces instructions SQL peuvent être issues d'activités de service d'information ou peuvent correspondre à des instructions exécutées lors du processus d'installation ou du démarrage d'une instance.

Configuration de la source de données d'une application de processus et des paramètres de référence d'ensemble

Il peut être nécessaire de configurer les applications de processus exécutant des instructions SQL pour une infrastructure de base de données spécifique. Ces instructions SQL peuvent être issues d'activités de service d'information ou peuvent correspondre à des instructions exécutées lors du processus d'installation ou du démarrage d'une instance.

task_context

Lorsque vous installez l'application, vous pouvez spécifier les types de sources de données suivants :

- Sources de données pour l'exécution d'instructions SQL lors de l'installation du processus
- Sources de données pour l'exécution d'instructions SQL lors du démarrage d'une instance de processus
- Sources de données pour l'exécution d'activités de fragments SQL

La source de données requise pour exécuter une activité de fragments SQL est définie dans une variable BPEL de type tDataSource. Le schéma de base de

données et les noms de table requis pour une activité de fragments SQL sont définis dans des variables BPEL de type tSetReference. Vous pouvez configurer les valeurs initiales de ces deux variables.

Vous pouvez spécifier les sources de données à l'aide de l'outil wsadmin.

task_procedure

1. Installez l'application de processus de manière interactive à l'aide de l'outil wsadmin.
2. Parcourez les tâches jusqu'à atteindre celles permettant de mettre à jour des sources de données et des références d'ensemble.

   Configurez ces paramètres pour votre environnement. L'exemple suivant présente les paramètres que vous pouvez modifier pour chacune de ces tâches.
3. Enregistrez vos modifications.

Exemple : Mise à jour de sources de données et des références d'ensemble à l'aide de l'outil wsadmin

Dans la tâche **Mise à jour des sources de données**, vous pouvez modifier les valeurs des sources de données par des valeurs de variables initiales utilisées lors de l'installation du processus ou au démarrage de ce dernier. Dans la tâche **Mise à jour des références d'ensemble**, vous pouvez configurer les paramètres liés au schéma de base de données et aux noms de table.

```
Task[24] : Mise à jour des sources de données

//Modifiez les valeurs des sources de données pour les variables initiales lors
du démarrage du processus

Nom du processus :  Test
// Nom du modèle de processus
Démarrage du processus ou heure d'installation : Process start
// Indique si la valeur spécifiée est évaluée
//lors du démarrage ou de l'installation du processus
Instruction ou variable :  Variable
// Indique qu'une variable de source de données doit être modifiée
Nom de la source de données :  MyDataSource
// Nom de la variable
Nom JNDI :[jdbc/sample] :jdbc/newName
// Définit le nom JNDI sur jdbc/newName
Task[25]: Mise à jour des références d'ensemble

// Modifiez les valeurs des références d'ensemble utilisées en tant
que valeurs initiales pour les variables BPEL

Nom du processus :  Test
// Nom du modèle de processus
Variable :  SetRef
// Nom de la variable BPEL
Nom JNDI :[jdbc/sample] :jdbc/newName
// Définit le nom JNDI de la source de données de référence de l'ensemble
sur jdbc/newName Nom du schéma : [IISAMPLE]
// Nom du schéma de la base de données
Préfixe de schéma :  [] :
// Préfixe du nom du schéma.
// Ce paramètre s'applique uniquement si le nom du schéma est généré.
Nom de table : [SETREFTAB] : NEWTABLE
// Définit le nom de la table de base de données sur NEWTABLE
Préfixe de table :  [] :
// Préfixe du nom de table.
// Ce paramètre s'applique uniquement si le nom de la table est généré.
```

Désinstallation d'applications de processus métier et de tâche manuelle à l'aide de la console d'administration

Vous pouvez utiliser la console d'administration pour désinstaller des applications contenant des processus métier ou des tâches manuelles.

task_prereq

Pour désinstaller une application contenant des processus métier ou des tâches manuelles, assurez-vous que les conditions suivantes sont remplies :

- Si l'application est installée sur un serveur autonome, le serveur doit être démarré et avoir accès à la base de données de Business Process Choreographer.
- Si l'application est installée sur un cluster, le gestionnaire de déploiement et au moins un membre du cluster doivent être en cours d'exécution. Le membre de cluster doit avoir accès à la base de données de Business Process Choreographer.
- Si l'application est installée sur un serveur géré, le gestionnaire de déploiement et ce serveur doivent être en cours d'exécution. Le serveur doit avoir accès à la base de données de Business Process Choreographer.
- Aucune instance de modèle de processus métier ou de tâche manuelle, quel que soit son état, n'est présente, ou vous avez un serveur autonome qui s'exécute en mode développement.
- Si une instance de processus a été migrée vers une version plus récente du processus, mais qu'elle attend qu'un appel de service réponde, l'application contenant la version précédente ne peut pas être désinstallée tant que la réponse n'est pas reçue. Dans tous les autres cas, les instances qui ont été migrées sont considérées comme des instances de la nouvelle version, et l'application contenant la version précédente du processus peut être désinstallée.

task_context

Pour désinstaller une application d'entreprise contenant des processus métier ou des tâches manuelles, effectuez les opérations suivantes :

task_procedure

1. Dans la console d'administration, cliquez sur **Applications** → **Types d'application** → **Applications d'entreprise WebSphere**.
2. Sélectionnez l'application à désinstaller et cliquez sur **Arrêter**.

   Cette étape échoue si des instances de processus ou de tâche existent toujours dans l'application. Vous pouvez soit utiliser Business Process Choreographer Explorer pour supprimer les instances, soit utiliser l'option **-force** du script d'administration bpcTamplates.jacl pour arrêter et supprimer ces instances avant la désinstallation de l'application.
3. Sélectionnez l'application à désinstaller et cliquez sur **Désinstaller**.
4. Cliquez sur **Sauvegarder** pour enregistrer les modifications.

task_results

L'application est désinstallée.

Tâches associées

«Désinstallation d'applications de processus métier et de tâches manuelles à l'aide d'une commande d'administration»
L'utilisation du script bpcTemplates.jacl est une alternative à l'emploi de la console d'administration pour désinstaller des applications qui contiennent des processus métier ou des tâches manuelles.

Désinstallation d'applications de processus métier et de tâches manuelles à l'aide d'une commande d'administration

L'utilisation du script bpcTemplates.jacl est une alternative à l'emploi de la console d'administration pour désinstaller des applications qui contiennent des processus métier ou des tâches manuelles.

task_prereq

Pour désinstaller une application contenant des processus métier ou des tâches manuelles, assurez-vous que les conditions suivantes sont remplies :

- Si l'application est installée sur un serveur autonome, le serveur doit être démarré et avoir accès à la base de données de Business Process Choreographer.
- Si l'application est installée sur un cluster, le gestionnaire de déploiement et au moins un membre du cluster doivent être en cours d'exécution. Le membre de cluster doit avoir accès à la base de données de Business Process Choreographer.
- Si l'application est installée sur un serveur géré, le gestionnaire de déploiement et ce serveur doivent être en cours d'exécution. Le serveur doit avoir accès à la base de données de Business Process Choreographer.
- Assurez-vous que le processus serveur auquel le client d'administration se connecte est en cours d'exécution. Pour vous assurer que le client d'administration se connecte automatiquement au processus serveur, ne spécifiez pas l'option `-conntype NONE` en tant qu'option de commande.
- Si la sécurité administrative WebSphere est activée et si votre ID utilisateur ne dispose pas des droits d'opérateur ou d'administrateur, incluez les options wsadmin `-user` et `-password` pour indiquer un ID utilisateur disposant des droits d'opérateur ou d'administrateur. L'option `-uninstall` requiert les droits d'opérateur et l'option `-force` les droits d'administrateur.
- Une ou plusieurs des conditions suivantes sont vérifiées :
  - Il n'existe pas d'instance de modèle de processus métier ou de tâche manuelle.
  - Vous voulez utiliser l'option **-force**.
  - Vous avez un serveur autonome qui s'exécute en mode développement.
- Si une instance de processus a été migrée vers une version plus récente du processus, mais qu'elle attend qu'un appel de service réponde, l'application contenant la version précédente ne peut pas être désinstallée tant que la réponse n'est pas reçue. Dans tous les autres cas, les instances qui ont été migrées sont considérées comme des instances de la nouvelle version, et l'application contenant la version précédente du processus peut être désinstallée.

task_context

Les étapes suivantes expliquent comment utiliser le script bpcTemplates.jacl pour désinstaller des applications contenant des modèles de processus métier ou de tâche manuelle.

task_procedure

1. Si des instances de processus ou de tâche sont encore associées aux modèles contenus dans l'application que vous comptez désinstaller, supprimez-les en appliquant l'une des procédures suivantes (ou les deux) :
   - Utilisez Business Process Choreographer Explorer pour supprimer les instances.
   - Dans les cas où vous êtes sûr qu'aucun autre processus métier ne dépend des modèles de processus qui sont définis dans l'application que vous souhaitez désinstaller, vous pouvez utiliser l'option **-force**.

     ATTENTION :
     Si vous utilisez le script avec cette option, il supprime toutes les instances qui sont associées aux modèles, toutes les données qui sont associées aux instances en cours d'exécution, il arrête les modèles et désinstalle l'application en une étape. Utilisez cette option avec précaution.
2. Sélectionnez le sous-répertoire de Business Process Choreographer dans lequel se trouvent les scripts d'administration. Entrez la commande suivante :

   ```
   cd racine_installation/ProcessChoreographer/admin
   ```

   Linux UNIX Sous Linux et UNIX, entrez la commande suivante :

   ```
   cd racine_installation/ProcessChoreographer/admin
   ```

   Sur la plateforme i5/OS, entrez la commande suivante :

   ```
   cd racine_installation/ProcessChoreographer/admin
   ```

   Windows Sous Windows, entrez la commande suivante :

   ```
   cd racine_installation\ProcessChoreographer\admin
   ```
3. Arrêtez les modèles et désinstallez l'application correspondante.

   Windows Sous Windows, entrez :

   ```
   racine_installation\bin\wsadmin -f bpcTemplates.jacl
                                    -uninstall nom_application
                                   [-force]
   ```

   Linux UNIX Sur les plateformes Linux et UNIX, entrez :

   ```
   racine_installation/bin/wsadmin -f bpcTemplates.jacl
                                    -uninstall nom_application
                                   [-force]
   ```

   Où :

   -uninstall *nom_application*
   Indique le nom de l'application à désinstaller.

   -force
   Cette option entraîne l'arrêt et la suppression des instances en cours d'exécution avant que l'application soit désinstallée. Utilisez cette option avec précaution car elle supprime également toutes les données associées aux instances en cours d'exécution.

task_results

L'application est désinstallée.

Tâches associées

«Désinstallation d'applications de processus métier et de tâche manuelle à l'aide de la console d'administration», à la page 584
Vous pouvez utiliser la console d'administration pour désinstaller des applications contenant des processus métier ou des tâches manuelles.

Chapitre 7. Adaptateurs et installation

Les adaptateurs permettent à votre application de communiquer avec d'autres composants du système d'information d'entreprise.

La procédure d'installation des adaptateurs est décrite dans la rubrique Configuration et utilisation des adaptateurs du centre de documentation de WebSphere Integration Developer.

Chapitre 8. Identification et résolution des incidents lors d'un échec de déploiement

Ce chapitre décrit les étapes nécessaires afin de déterminer la cause d'un problème survenu lors du déploiement d'une application. Il présente également des solutions possibles.

task_prereq

Cette rubrique suppose que les conditions suivantes sont remplies :
- Vous comprenez les principes de base du débogage d'un module.
- Les fonctions de journalisation et de trace sont actives pendant le déploiement du module.

task_context

La tâche de résolution des incidents de déploiement commence lorsque vous recevez une notification d'erreur. Lors d'un échec de déploiement, il existe divers symptômes que vous devez inspecter avant d'agir.

task_procedure

1. Déterminez si l'installation de l'application a échoué.

   Cherchez dans le fichier `SystemOut.log` des messages qui indiquent la cause de l'échec. Les raisons de l'échec de l'installation d'une application peuvent être notamment les suivantes :

   - Vous essayez d'installer une application sur plusieurs serveurs dans la même cellule Network Deployment.
   - Une application possède le même nom qu'un module existant de la cellule Network Deployment dans laquelle vous installez l'application.
   - Vous essayez de déployer des modules Java EE dans un fichier EAR sur différents serveurs cible.

   Important : Si l'installation a échoué et que l'application contient des services, vous devez supprimer toutes les destinations SIBus ou les spécifications d'activation JCA créées avant l'échec et avant la tentative de réinstallation de l'application. Le moyen le plus simple de supprimer ces artefacts est de cliquer sur **Sauvegarder -> Annuler tout** après l'échec. Si vous enregistrez par inadvertance les modifications, vous devez supprimer manuellement les destinations SIBus destinations et les spécifications d'activation JCA (voir les rubriques concernant la suppression des destinations SIBusand et les spécifications d'activation JCA, à la section Administration).
2. Si l'application est installée correctement, examinez-la pour déterminer si elle a été démarrée avec succès.

   Si le démarrage de l'application a échoué, l'échec s'est produit lorsque le serveur a tenté d'initier les ressources de l'application.

   a. Cherchez dans le fichier `SystemOut.log` des messages qui vous indiquent comment continuer.

   b. Déterminez si les ressources requises par l'application sont disponibles et/ou si leur démarrage a réussi.

Les ressources qui n'ont pas démarré empêchent une application de s'exécuter. Cela empêche la perte d'informations. Les raisons pour lesquelles une ressource ne démarre pas incluent :
- Les liaisons sont spécifiées de manière incorrecte
- Les ressources sont configurées de manière incorrecte
- Les ressources ne se trouvent pas dans le fichier RAR (fichier archive de ressources)
- Des ressources Web ne se trouvent pas dans le fichier WAR (fichier archive de services Web)

c. Déterminez si des composants sont manquants.

La raison de l'absence d'un composant est un fichier EAR mal compilé. Assurez-vous que tous les composants requis par le module se trouvent dans les dossiers appropriés du système test sur lequel vous avez compilé le fichier JAR (archive Java). «Préparation du déploiement sur un serveur» contient des informations supplémentaires.

3. Regardez si des informations circulent dans l'application.

Même une application en cours d'exécution peut rencontrer un échec lors du traitement des informations. Les raisons de ce problème sont similaires à celles qui sont mentionnées à l'étape 2b, à la page 589.

a. Déterminez si l'application utilise des services contenus dans une autre application. Vérifiez que l'autre application est installée et a démarré avec succès.

b. Déterminez si les liaisons d'importation et d'exportation de tous les services contenus dans d'autres applications utilisées par l'application défaillante sont configurées correctement. Utilisez la console d'administration pour examiner et corriger les liaisons.

4. Corrigez le problème et relancez l'application.

Tâches associées

«Suppression des spécifications d'activation JCA»
Le système génère des spécifications d'application JCA lors de l'installation d'une application contenant des services. Dans certains cas, vous devez supprimer ces spécifications avant de réinstaller l'application.

«Suppression des destinations SIBus», à la page 591
Les destinations de bus d'intégration de services (SIbus) contiennent les messages en cours de traitement au niveau des modules SCA. En cas d'incident, il peut être nécessaire de supprimer des destinations de bus pour résoudre le problème.

Suppression des spécifications d'activation JCA

Le système génère des spécifications d'application JCA lors de l'installation d'une application contenant des services. Dans certains cas, vous devez supprimer ces spécifications avant de réinstaller l'application.

task_prereq

Si vous supprimez la spécification en raison de l'échec de l'installation d'une application, assurez-vous que le nom JNDI (Java Naming and Directory Interface) du module correspond au nom du module dont l'installation a échoué. La seconde partie du nom JNDI correspond au nom du module qui a implémenté la destination. Par exemple, dans `sca/SimpleBOCrsmA/ActivationSpec`, **SimpleBOCrsmA** correspond au nom du module.

Rôle de sécurité requis pour cette tâche : Lorsque la sécurité et les autorisations par rôle sont activées, vous devez être connecté en tant qu'administrateur ou configurateur pour exécuter cette tâche.

task_context

Supprimez les spécifications d'activation JCA lorsque vous enregistrez par mégarde une configuration après avoir installé une application qui contient des services et ne nécessite aucune spécification.

task_procedure

1. Localisez la spécification d'activation à supprimer.

   Les spécifications sont contenues dans le panneau relatif aux adaptateurs de ressources. Accédez à ce panneau en cliquant sur **Ressources > Adaptateurs de ressources**.

   a. Localisez l'**adaptateur de ressources SPI du composant de messagerie de plateforme**.

      Pour cela, vous devez vous placer au niveau du **noeud** pour un serveur autonome ou au niveau du **serveur** pour un environnement de déploiement.
2. Affichez les spécifications d'activation JCA associées à l'adaptateur de ressources SPI du composant de messagerie de plateforme.

   Cliquez sur le nom de l'adaptateur de ressources, un panneau répertoriant les spécifications associées s'affiche.
3. Supprimez toutes les spécifications dont le **Nom JNDI** correspond à celui du module que vous avez supprimé.

   a. Cochez la case située en regard de chacune des spécifications concernées.

   b. Cliquez sur **Supprimer**.

task_results

Le système supprime les spécifications sélectionnées de l'écran d'affichage.

task_postreq

Sauvegardez les modifications.

Tâches associées

«Suppression des destinations SIBus»
Les destinations de bus d'intégration de services (SIbus) contiennent les messages en cours de traitement au niveau des modules SCA. En cas d'incident, il peut être nécessaire de supprimer des destinations de bus pour résoudre le problème.

Suppression des destinations SIBus

Les destinations de bus d'intégration de services (SIbus) contiennent les messages en cours de traitement au niveau des modules SCA. En cas d'incident, il peut être nécessaire de supprimer des destinations de bus pour résoudre le problème.

task_prereq

Si vous supprimez la destination en raison de l'échec de l'installation d'une application, assurez-vous que le nom du module de la destination correspond au nom du module dont l'installation a échoué. La seconde partie du nom de la

destination correspond au nom du module qui a implémenté la destination. Par exemple, dans `sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer`, **SimpleBOCrsmA** correspond au nom du module.

Rôle de sécurité requis pour cette tâche : Lorsque la sécurité et les autorisations par rôle sont activées, vous devez être connecté en tant qu'administrateur ou configurateur pour exécuter cette tâche.

task_context

Supprimez les destinations SIBus lorsque vous enregistrez par mégarde une configuration après avoir installé une application qui contient des services et n'avez plus besoin des destinations.

Remarque : Cette tâche supprime la destination du bus système SCA uniquement. Vous devez également supprimer les entrées du bus d'application avant de réinstaller une application qui contient des services (voir la rubrique Suppression des spécifications d'activation JCA dans la section relative à l'administration de ce centre de documentation).

task_procedure

1. Connectez-vous à la console d'administration.
2. Affichez les destinations sur le bus système SCA.
   a. Dans la sous-fenêtre de navigation, cliquez sur **Intégration de service → bus**
   b. Dans la sous-fenêtre de contenu, cliquez sur **SCA.SYSTEM.*nom_cellule*.Bus**
   c. Dans Ressources de destination, cliquez sur **Destinations**
3. Cochez la case en regard de chaque destination associée à un nom du module correspondant au module en cours de suppression.
4. Cliquez sur **Supprimer**.

task_results

Le panneau affiche uniquement les destinations restantes.

task_postreq

Supprimez les spécifications d'activation JCA associées au module qui a créé ces destinations.

Tâches associées

«Suppression des spécifications d'activation JCA», à la page 590
Le système génère des spécifications d'application JCA lors de l'installation d'une application contenant des services. Dans certains cas, vous devez supprimer ces spécifications avant de réinstaller l'application.