WebSphere Process Server for z/OS

テクニカル概説

バージョン 7.0.0

本書は、WebSphere Process Server for z/OS バージョン 7、リリース 0、モディフィケーション 0 (製品番号 5655-N53)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本書についてのご意見は、doc-comments@us.ibm.com へ E メールでお寄せください。皆様の率直なご意見をお待ちしています。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： WebSphere Process Server for z/OS
Technical Overviews
Version 7.0.0

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第2刷 2012.5

目次

テクニカル概説

テクニカル概説情報では、製品体系に関連する標準および技術的側面を紹介します。

Service Component Architecture

Service Component Architecture (SCA) は、サービス指向アーキテクチャーを使用可能にするもので、IBM® を含む多くの企業が提供しています。SCA は、プラットフォームおよびベンダーから独立したプログラミング・モデルで、技術的な実装の詳細に関わらず、ビジネス・ロジックとビジネス・データを SOA サービスとして表現するためのシンプルで一貫性のある手段を提供します。このセクションでは、SCA サービスとデータ・オブジェクトについて検討します。

SCA およびサービス呼び出し

データ、呼び出し、構成というプログラミング・モデルの 3 つの側面を捉え、サービスをベースとした手法の新しいパラダイムのいくつかを適用すると、SOA の新たなプログラミング・モデルが見えてきます。Service Component Architecture (SCA) は、SOA ソリューション内でビジネス・サービスを呼び出す手段を提供します。

サービス指向アーキテクチャーを構成するアーキテクチャー構成体には、サービス間で交換されるデータを表現する方法、サービスを呼び出すメカニズム、およびサービスから大規模統合ビジネス・アプリケーションを構成する方法が含まれています。現在は、これらのそれぞれをサポートするための多数の異なるプログラミング・モデルがあります。この状況により、開発者には、特定のビジネス上の問題を解決するだけでなく、適切な実装テクノロジーを選択して理解するという課題も与えられています。WebSphere® Process Server SOA ソリューションの重要な目的の 1 つは、このような複雑さを緩和することです。これは、サービス指向ビジネス・アプリケーションを実装するために使用されるさまざまなプログラミング・モデルを 1 つにまとめて、簡素化されたプログラミング・モデルにすることによって、実現します。

このセクションでは特に、ビジネス・サービスを定義して呼び出すサービス指向コンポーネント・モデルとしての WebSphere Process Server 内の Service Component Architecture (SCA) に焦点を当てています。SCA は、WebSphere Process Server の SOA ソリューションに呼び出しモデルを提供するために重要な役割を果たしています。SCA は、ビジネス・サービスから複合ビジネス・アプリケーションを構成する場合にも役割を果たします。

第一に分かるのは、データは主に Extensible Markup Language (XML) によって表されており、サービス・データ・オブジェクト (SDO) 仕様に基づくビジネス・オブジェクトを使用するか、または XPath や XSLT (Extensible Stylesheet Language Transformation) などのネイティブ XML 機能を介してプログラムされることです。第二に、サービス呼び出しは Service Component Architecture (SCA) にマップされます。最後に、Business Process Execution Language (BPEL) を使用したプロセス・オ

ーケストレーションのなかに構成が統合されます。以下の図に、この新しいプログラミング・モデルの 3 つの側面を示します。

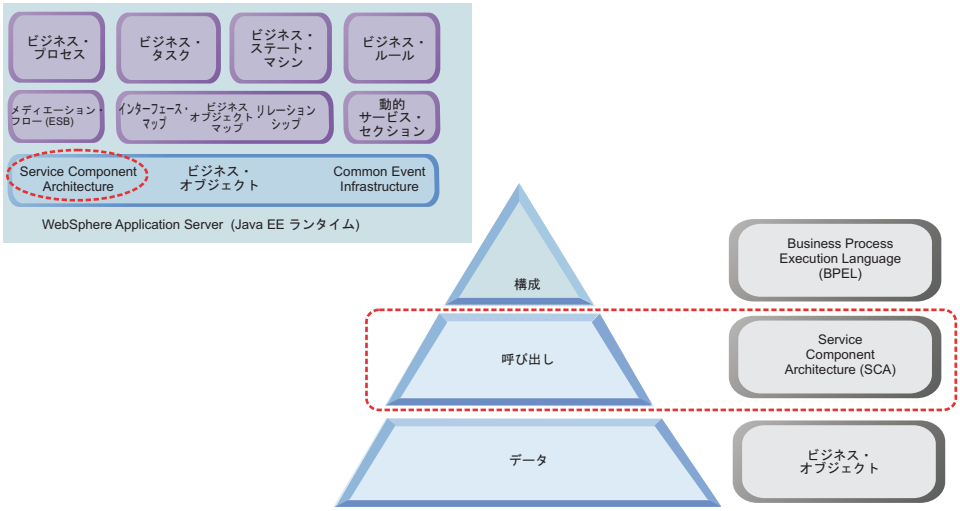


図 1. SOA のプログラミング・モデル内部のデータ、呼び出し、および構成の表現

SCA は、Java EE ランタイム環境で実行されるアプリケーションを作成するための簡素化されたプログラミング・モデルを提供することを目的としており、既存の Java EE テクノロジーを改良した概念および技法に基づいています。SCA の重要な性質の 1 つは、アプリケーション・ビジネス・ロジックを、実装の詳細と分離できるようにしている点です。これを実現するため、SCA では、セッション Bean、Web サービス、Java クラス、または BPEL として既に表現されている可能性のあるサービス・タイプを 1 つに抽象化できます。ビジネス・ロジックをインフラストラクチャー・ロジックから分離できることは、エンタープライズ・アプリケーションを構築するために必要な IT リソースを削減し、開発者が、どの実装テクノロジーを使用すべきかについての詳細に重点を置くのではなく、特定のビジネス上の問題に取り組むための時間を増やすために重要です。

モジュール

モジュール は、エンタープライズ・アーカイブ (EAR) ファイルに一緒にパッケージされる成果物を決定するデプロイメントの単位です。モジュール内のコンポーネントは、パフォーマンスを考慮して連結され、参照によってデータを渡すことができます。モジュールは、スコープを設定するメカニズムと見なすことができます。つまり、モジュールによって成果物の組織上の境界が設定されます。

モジュールは、サービス・コンポーネント、インポート、およびエクスポートの複合体です。サービス・コンポーネント、インポート、およびエクスポートは同じプロジェクトのルート・フォルダーに存在します。ここには、インポートおよびエクスポートに必要なバインディングとコンポーネントをリンクするワイヤリングも含まれます。モジュールには、そのコンポーネント、インポート、およびエクスポートによって参照される実装とインターフェースも含まれていることがあります。あるいは、これらの実装とインターフェースが、ライブラリー・プロジェクトなど、他のプロジェクトに配置されることもあります。

モジュールには 2 つのタイプがあります。一つは、*モジュール* と呼ばれるモジュールです (ビジネス・インテグレーション・モジュールと呼ばれることもあります)。これには、多数のコンポーネント・タイプから選択したものが含まれており、ビジネス・プロセスのサポートによく使用されます。もう一つは、*メディエーション・モジュール* と呼ばれるモジュールです。これには最大で 1 つのコンポーネント、1 つ以上のメディエーション・フロー・コンポーネント、およびメディエーション・フロー・コンポーネントを拡張する 0 個以上の Java コンポーネントが含まれます。

モジュールには、1 つ以上のメディエーション・フロー・コンポーネントを格納することができます。

なぜ 2 つのモジュール・タイプがあるのでしょうか。最初のタイプのモジュールは、主にビジネス・プロセス用に設計されています。メディエーション・モジュールは、既存の外部サービスへのゲートウェイのようなものであり、エンタープライズ・サービス・バス・アーキテクチャーでよく使用されます。これらの外部サービスまたはエクスポートは、インポートまたはサービス・プロバイダーによってメディエーション・モジュール内でアクセスされます。メディエーション・フローによってクライアント・サービス要求元をサービス・プロバイダーから分離することにより、アプリケーションの柔軟性および回復力が増し、サービス指向アーキテクチャーという目標に到達できます。例えば、メディエーション・フローでは、着信メッセージをログに記録したり、実行時に決定された特定のサービスにメッセージを送信したり、データを別のサービスに渡せるように変換したりできます。これらの機能は、要求元またはプロバイダー・サービスを変更せずに、長期的に追加および変更することができます。

モジュールにより、サービス・アプリケーションがテストされ、WebSphere Process Server にデプロイされます。メディエーション・モジュールにより、サービス・アプリケーションがテストされ、WebSphere Process Server または WebSphere Enterprise Service Bus サーバーのいずれかにデプロイされます。どちらのタイプのモジュールでも、インポートおよびエクスポートがサポートされます。

実装、インターフェース、ビジネス・オブジェクト、ビジネス・オブジェクト・マップ、ロール、リレーションシップ、およびその他の成果物は、多くの場合、モジュール間で共有する必要があります。*ライブラリー* は、これらの共有リソースの保管に使用されるプロジェクトです。

4 ページの図 2 では、モジュールに 2 つのサービス・コンポーネントが含まれており、それぞれに実装が含まれています。モジュールには、サービス・コンポーネントが必要とする適切なインターフェースおよび参照も含まれます。2 番目のサービス・コンポーネントは、外部サービスを呼び出さないため、参照は含まれていません。

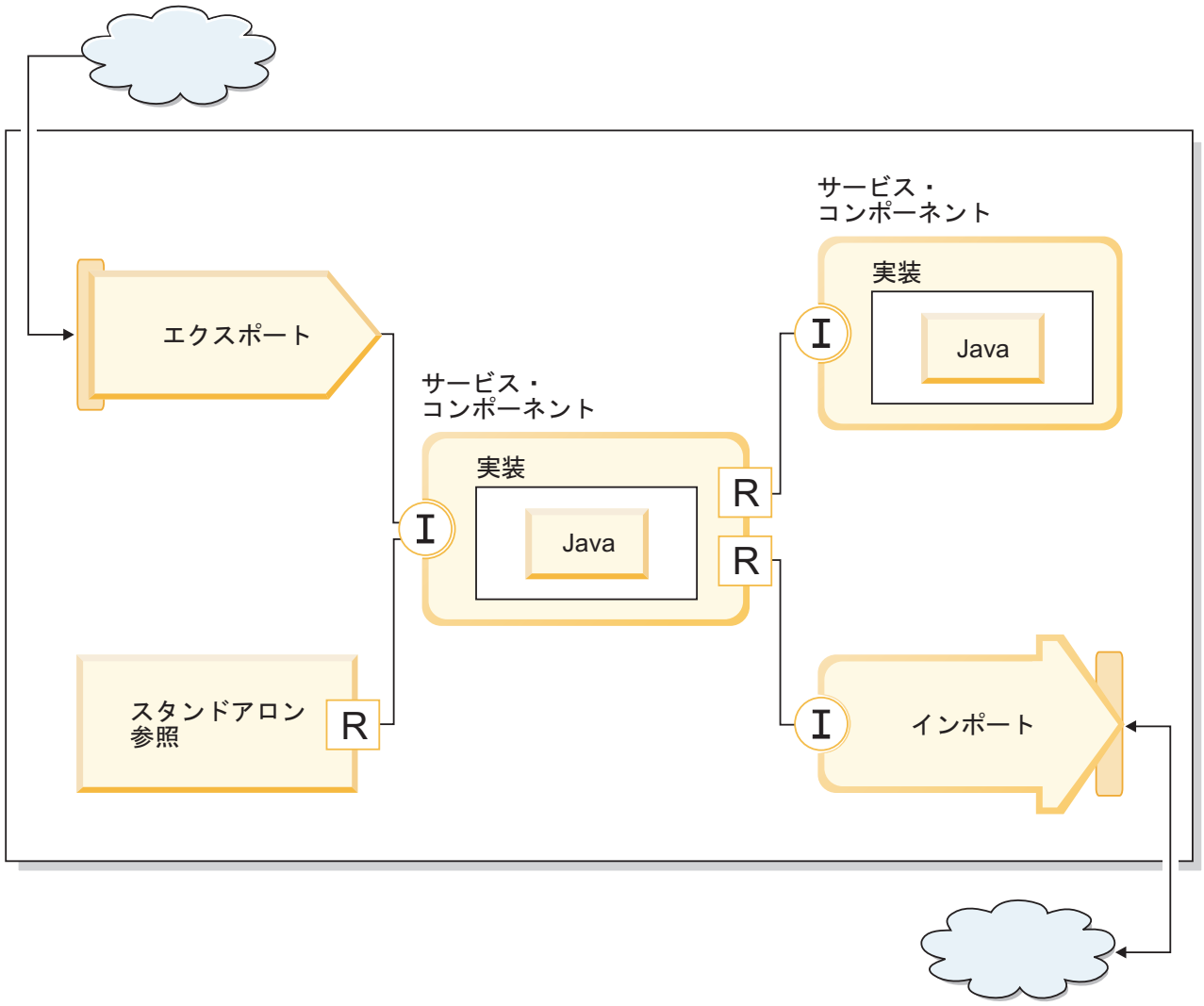
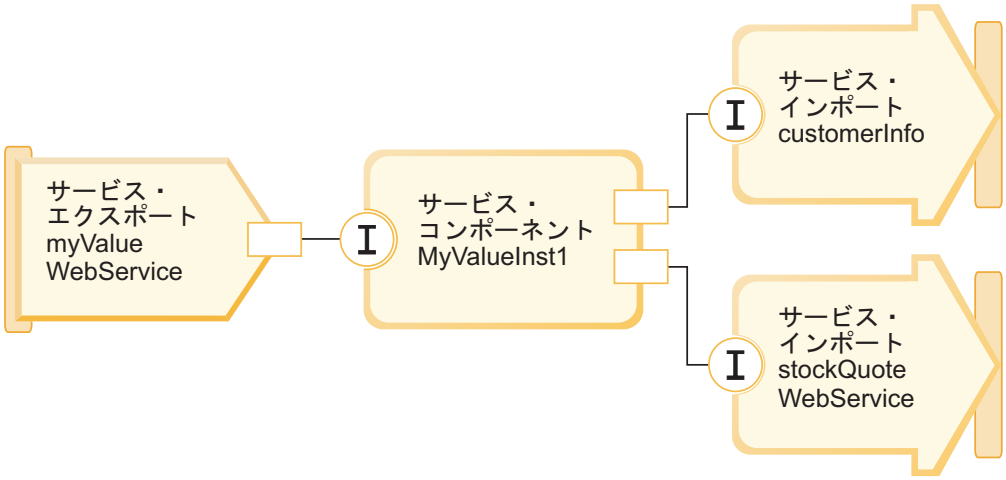


図 2. モジュールの構造

図 3 では、モジュールに 1 つのエクスポート、2 つのインポート、およびそれらを使用するサービス・コンポーネントが含まれています。インターフェースと参照をリンクするワイヤリングが示されています。

図 3. サービス・モジュール



モジュールおよびメディエーション・モジュールの成果物としては、以下のものがあります。

- モジュール定義 - モジュールの定義。
- サービス・コンポーネント - モジュール内のサービスの定義。モジュール内のサービス・コンポーネント名は固有です。ただし、サービス・コンポーネントには任意の表示名を付けることができます。一般に、表示名はユーザーにとって分かりやすい名前にします。
- インポート - インポートの定義。これは、このモジュールにとって外部のサービスの呼び出しです。インポートにはバインディングが含まれます。これについては、バインディングのセクションで説明します。
- エクスポート - エクスポートの定義。このモジュールにとって外部の呼び出し元にコンポーネントを公開するために使用されます。エクスポートにはバインディングが含まれます。これについては、バインディングのセクションで説明します。
- 参照 - モジュール内のあるコンポーネントから別のコンポーネントへの参照です。
- スタンドアロン参照 - Service Component Architecture コンポーネントとして定義されていないアプリケーション (例えば JavaServer Pages など) を参照します。これにより、これらのアプリケーションは Service Component Architecture コンポーネントと対話できるようになります。1 つのモジュールに存在することができるスタンドアロン参照成果物は 1 つだけです。
- その他の成果物 - これらの成果物としては WSDL ファイル、Java クラス、XSD ファイル、BPEL プロセスなどがあります。

サービス・コンポーネント

サービス・コンポーネントは、サービス実装を構成します。サービス・コンポーネントを、標準的なブロック・ダイアグラムで示します。

Service Component Architecture (SCA) はサービス呼び出し用の一貫性のある構文とメカニズムを提供するだけでなく、呼び出しフレームワークとして、開発者がサービス実装を再使用可能コンポーネントにカプセル化できる方法を提供します。SCA を使用すると、開発者は、使用されているテクノロジーに依存しない方法で、イン

ターフェース、実装、および参照を定義できます。この手法により、選択したテクノロジーの種類を問わず、要素をバインドする機会が得られます。SCA はビジネス・ロジックをインフラストラクチャーから分離し、アプリケーション・プログラマーがビジネス問題の解決に集中できるようにします。

コンポーネントは、実装 (WebSphere Integration Developer のツールを使用するときは非表示)、1 つ以上のインターフェース (入力、出力、および障害を定義する)、および 0 個以上の参照で構成されています。参照は、このコンポーネントが必要とするかまたはコンシュームする、別のサービスまたはコンポーネントのインターフェースを識別します。インターフェースは、WSDL ポート・タイプと Java の 2 つの言語のいずれかで定義できます。インターフェースは、同期対話スタイルと非同期対話スタイルをサポートします。コンポーネントの実装は、さまざまな言語で行うことができます。

推奨されるインターフェース・タイプは WSDL であり、チュートリアルおよびサンプルでは、一貫して WSDL インターフェース・タイプを使用します。ただし、Java インターフェースもサポートされており、主にステートレス・セッション EJB がインポートされる場合に使用されます。トップダウン Java コンポーネントを開発する場合、すなわち、コンポーネントを定義して後から Java 実装を追加する場合でも、WSDL インターフェースを使用する必要があります。WSDL インターフェースをベースとするコンポーネントを、Java インターフェースをベースとするコンポーネントと混用することはできません。

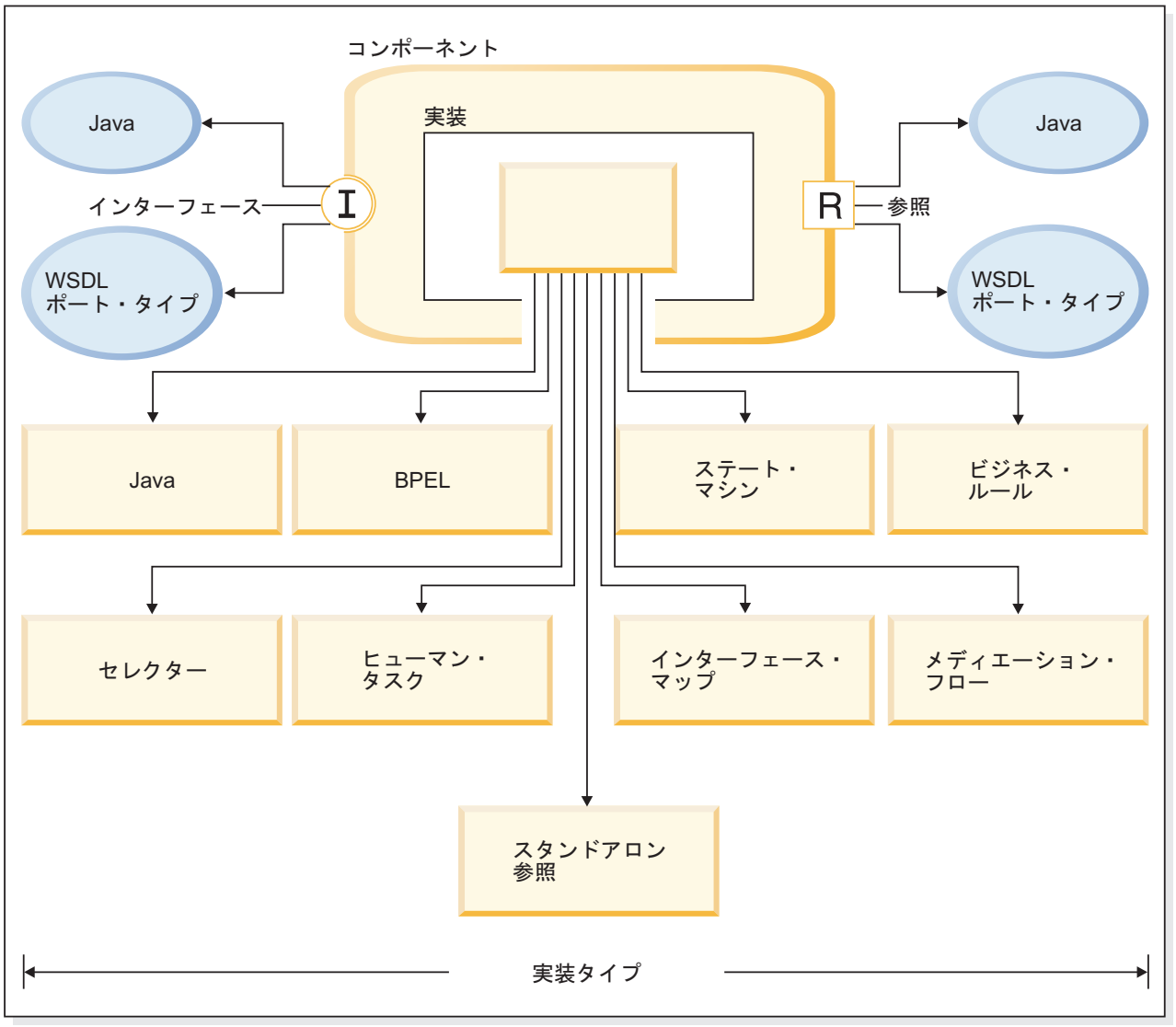


図 4. コンポーネントの構造

8 ページの図 5 では、中央にコンポーネントがあります。その実装である MyValueImpl は、インターフェースと同様に Java で書かれています。参照が 2 つあります。それらは、もう 1 つの Java インターフェースおよび WSDL インターフェースです。

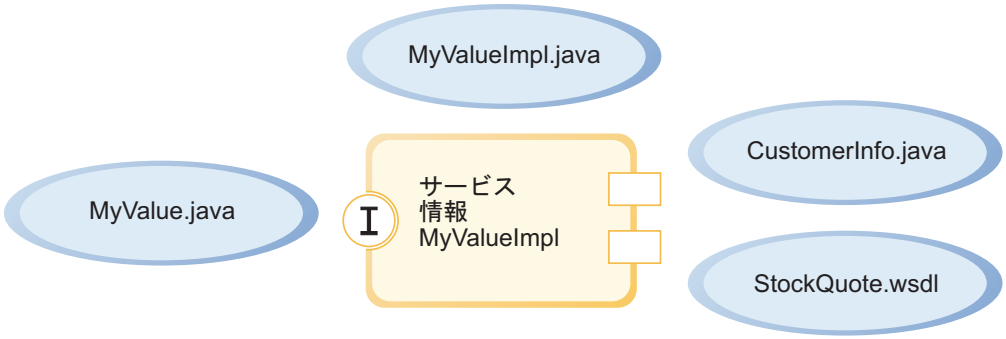


図 5. サービス・コンポーネント構造

このコンポーネントを処理するときに見えるのは、以下に示すように、実質的にコンポーネント自体のみです。別のコンポーネントからこのコンポーネントへの参照は、そのインターフェースへの線によって視覚的に示されます。このコンポーネントからの参照は、コンポーネントの参照点から他のコンポーネントのインターフェースへの線によって示されます。参照は、このコンポーネントがコンシュームするサービスを表します。参照の名前を示し、インターフェースのみを指定すると、コンポーネント実装作成者は、実際のサービスを参照するバインディングを先送りして後から実行することができます。後の時点で、統合スペシャリストが、別のコンポーネントまたはインポートのインターフェースへの参照からワイヤリングすることによって、バインディングを行います。バインディングの先送りと実装の再利用を可能にするこの疎結合は、WebSphere Integration Developer の Service Component Architecture を使用する主な理由の 1 つです。

コンポーネントには、プロパティーおよび修飾子が含まれる場合があります。修飾子は、ランタイム用のインターフェースおよび参照に対するサービス品質 (QoS) ディレクティブです。

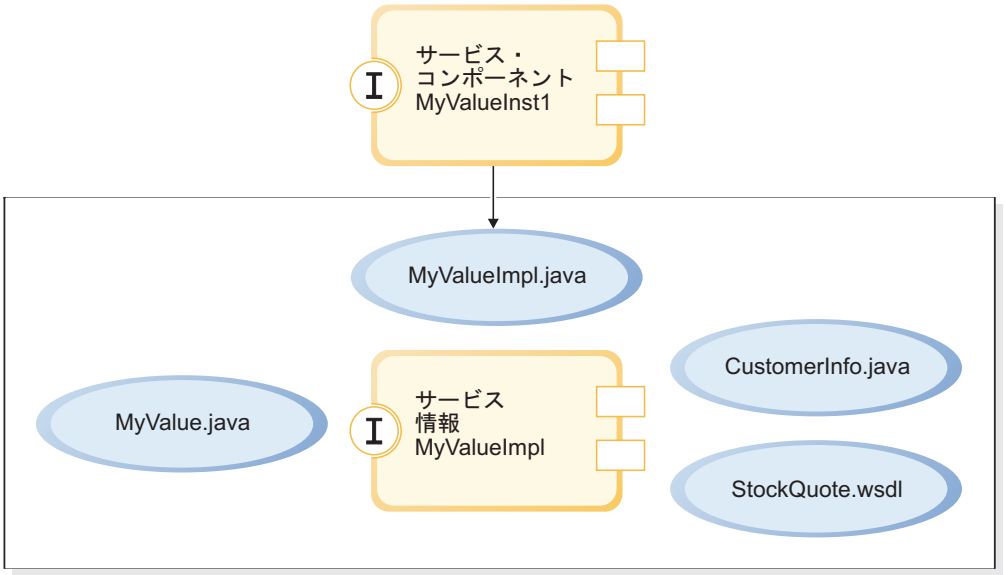


図 6. サービス・コンポーネントのインスタンス

サービス・コンポーネント実装タイプは、サービス・コンポーネントの実装です。

WebSphere Integration Developer は、WebSphere Process Server および WebSphere Enterprise Service Bus の以下の実装成果物をサポートします。

表 1. 実装成果物

WebSphere Process Server	**WebSphere Enterprise Service Bus**
Java オブジェクト	Java オブジェクト
ビジネス・プロセス	メディエーション・フロー
ビジネス・ステート・マシン	
ビジネス・ルール	
セレクター	
ヒューマン・タスク	
インターフェース・マップ	
メディエーション・フロー	

注: 注: インターフェース・マップは、WebSphere Process Server バージョン 7.0 では推奨されていません。WebSphere Integration Developer 内の既存のインターフェース・マップ・コンポーネントを、メディエーション・フロー・コンポーネント内の機能を使用するようにマイグレーションすることができます。

サービスの標準的なコンポーネントの実装について、このセクションのトピックで説明します。これらの実装は、アセンブリー・エディター内または BPEL プロセス内のサービスで見られます。

Java オブジェクト

Java でコンポーネントを実装したものは、Java オブジェクトと呼ばれます。

一般的な実装としては、Java で書かれたコンポーネントが挙げられます。この実装は、「Plain Old Java Object」(POJO) と呼ばれることもあります。一般に、この実装は WSDL インターフェース・タイプですが、この実装に Java インターフェースを持たせることも可能です。複数のインターフェースが指定されている場合は、WSDL インターフェースと Java インターフェースを混用することはできません。ただし、WSDL インターフェースのセットを使用して作成したアプリケーションを、Java インターフェースのセットを使用したアプリケーションに「結合」させることができます。「ようこそ」ビューのサンプル・ギャラリーにリストされたサンプルで、その方法を示しています。

Java オブジェクトを操作するとき、エディターのコンテキスト内でコードは非表示のままとなっています。

Java オブジェクトは、メディエーション・モジュールで使用できます。WebSphere Process Server または WebSphere Enterprise Service Bus サーバーのいずれかにデプロイすることができます。

BPEL プロセス

*BPEL プロセス・コンポーネント*は、ビジネス・プロセスを実装します。

その実装言語は、業界標準の Business Process Execution Language for Web Services (BPEL4WS) および IBM 拡張です。BPEL プロセスは、長期実行される可能性のあ

るサービスを、より単純なサービスを使用して実装します。プロセス・エディターで作成された BPEL プロセスは、以下の処理を実行できます。

- 制御フロー・グラフを使用して、他のサービスのオーケストレーションを記述する。
- 変数を使用してプロセス状態を保持する。
- 障害の処理を通じて高度なエラー処理を使用する。
- 非同期イベントをサポートする。
- 相関セットを使用して、インスタンスを識別する要求内部のビジネス・データ (例えば顧客 ID など) にマークを付けることにより、インバウンド要求を、特定のプロセスの正しいインスタンスに相関させる。
- 高度な補正サポートを通じて拡張トランザクションを提供する。

これらの標準的な BPEL 項目に加えて、WebSphere Integration Developer は、ヒューマン・タスク をサポートするプロセスに担当者を組み込むように BPEL を拡張します。例えば、この拡張では、人がローンを承認するという要件をプロセスに追加できます。

プロセス・エディターは、BPEL 構成体のビジュアル表示を使用して、ビジネス・プロセスを短時間で簡単に作成します。

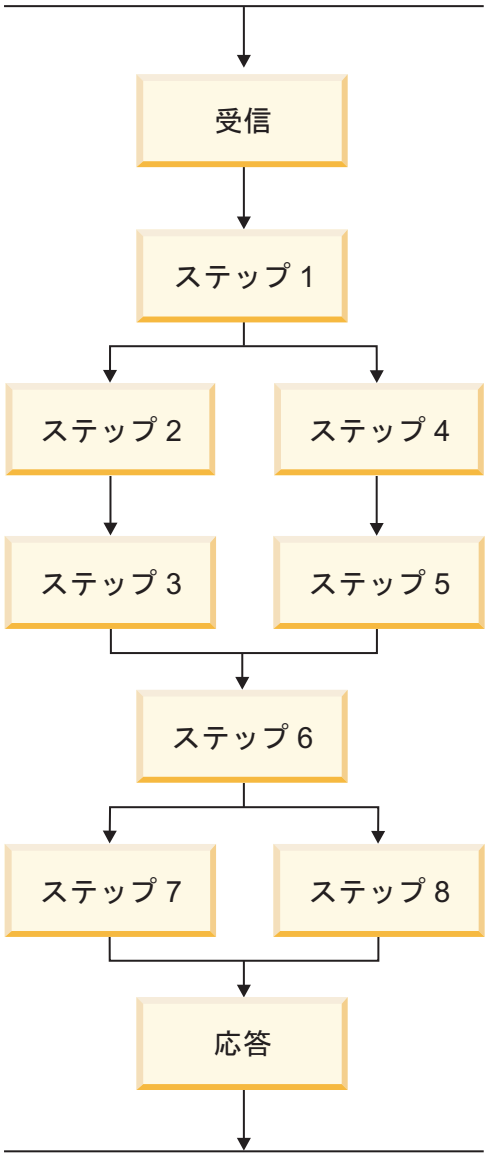


図 7. 単純なビジネス・プロセス

BPEL プロセスは、メディエーション・モジュールでは使用できません。WebSphere Process Server のみにデプロイできます。

ステート・マシン

ステート・マシンは、ビジネス・プロセスを作成するための別の方法です。ステート・マシンは、制御のフローではなく、状態を変更することに関連したプロセスに適しています。状態は、ある時点で成果物が実行できることを明示します。ステート・マシン は、この一連の状態を実装したものです。

ステート・マシンは、プロセス内の相互に関連した一連の状態を表示する一般的な方法です。よく知られているステート・マシンに、飲料自動販売機があります。自動販売機に硬貨を入れると、希望する飲み物と一緒に、ステート・マシンが投入された硬貨に基づいて機械的に釣り銭を用意するので、正確な釣り銭が出てきます。

以下のダイアグラムには、ステート・マシン・エディターによって作成される一般的なステート・マシンが示されています。このステート・マシンでは、商品が購入され、顧客に配送されます。

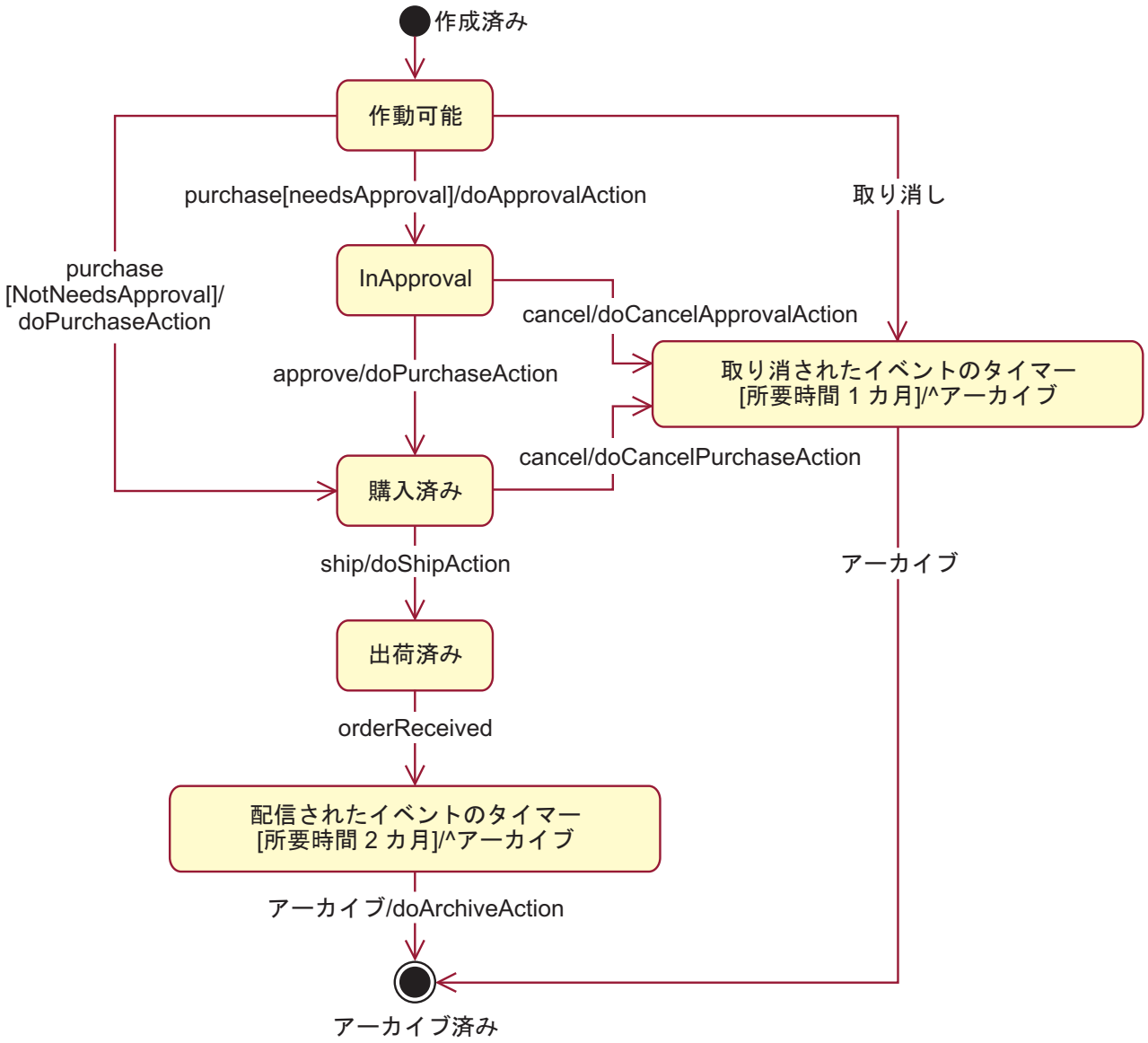


図 8. ステート・マシン

ステート・マシンは、メディエーション・モジュール内で使用することはできません。WebSphere Process Server のみにデプロイできます。

ビジネス・ルール

ビジネス・ルールは、ビジネス・プロセスおよび状態マシンを補足します。例えば、変数を使用する条件がある場合、*ビジネス・ルール* は、実行時にその変数の値を変更できます。ビジュアル・プログラミング言語によって作成されるビジネス・ルールは、コンテキストに基づいて決定を行います。その決定は単純な場合も複雑な場合もあります。ビジネス・ルールは非手続き型で、ルールはアプリケーションに関わりなく変更できます。

ビジネス・ルールは、コンテキストに基づくプロセスの結果を決定します。ビジネス・ルールは、特定の一連の状況で決定を下すために、日常のビジネス・シチュエーションで使用されます。この決定では、すべての状況を対象とするために多くのルールが必要となる場合があります。ビジネス・プロセス内のビジネス・ルールにより、アプリケーションはビジネス条件の変化に迅速に対応できます。例えば、保険会社では、申込者に対して自動車保険を承認するためのビジネス・ルールを、*申込者が男性で 25 歳を超えており、自動車カテゴリーがスポーツであり、当社と過去 5 年間保険契約を結んでいる場合、月ごとに $100 の保険料で保険の申し込みを承認する* という内容にすることができます。

WebSphere Integration Developer では、いくつかの方法でビジネス・ルールを作成できます。if-then ルールまたはデシジョン・テーブルを作成することができ、これらすべてがプロセスの結果を決定します。これらのルールはプロセス自体から独立しているので、プロセスを再実行せずに、いつでもルールを変更できます。例えば、ビジネスが置かれている状況に基づいて、*日付が 12 月 26 日から 1 月 1 日までの間である場合は、連休後セールとして 20% 割り引く* というルールを設定します。ただし、売上が低調な状態が続くようであれば、いつでも割引率を 40% に変更できます。

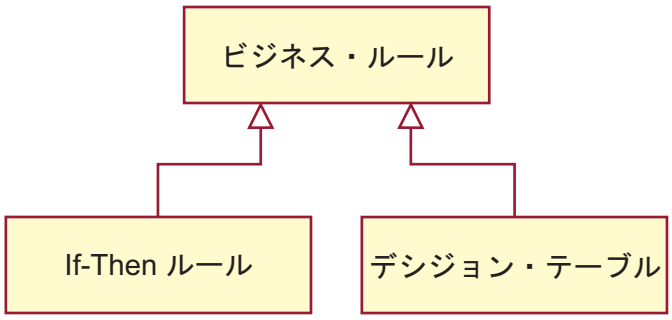


図 9. ビジネス・ルールのタイプ

ビジネス・ルールは、メディエーション・モジュールでは使用できません。WebSphere Process Server のみにデプロイできます。

セレクター

統合アプリケーションには、多くの対話方法が含まれています。オペレーションを、クライアント・アプリケーションから実装のいくつかの使用可能なコンポーネントのいずれかに経路指定するために、*セレクター* が使用されます。

コンポーネントへの経路指定は、日付に基づいています。例えば、*学校が始まる 2 週間前に、学校に関連する商品の新学期前の特別価格を提供する* という、日付に基づく経路があるとします。企業では、このように日付に基づく多くの経路を使用します。セレクターは、実行時に、日付に基づいて 1 つの経路を選択することを決定します。例えば、学校が始まる直前の場合、以前に新学期前に提供した価格が呼び出されます。ただし、学校が終わる時期の場合は、子供の夏休み向け商品の割引価格を設定する可能性があります。

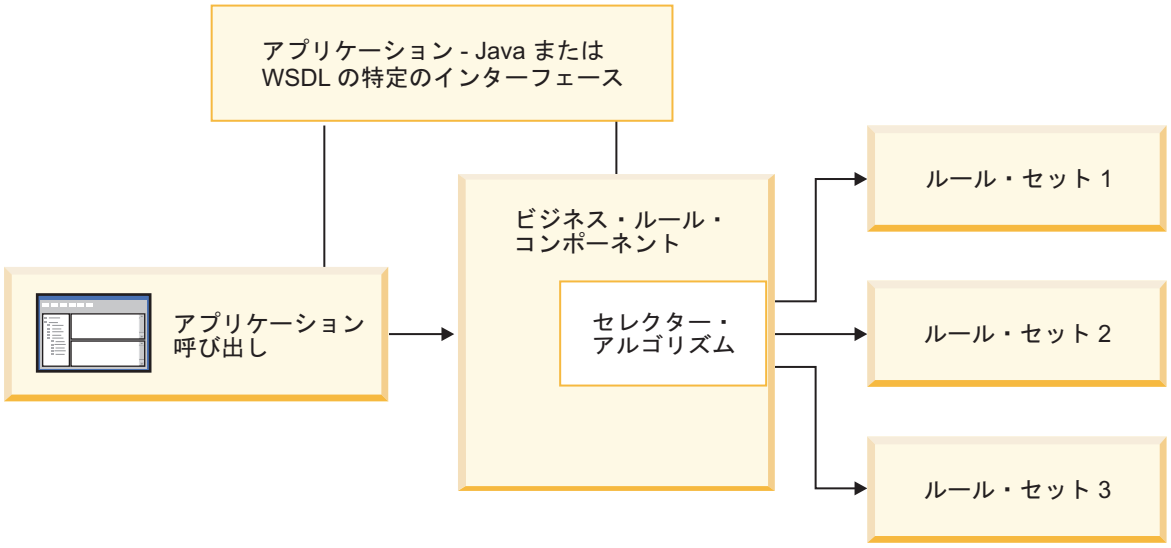


図 10. 一連のビジネス・ルールからの選択

セレクターは、メディエーション・モジュールでは使用できません。WebSphere Process Server のみにデプロイできます。

ヒューマン・タスク

ヒューマン・タスク ・コンポーネントは、人が実行するタスクを実装するものです。ビジネス・プロセスに人が関わることを表します。

時には、人がビジネス・プロセスに介入する必要があります。例えば、顧客が自分のクレジット限度額を超える品目を購入したい場合があります。ヒューマン・タスクを使用すると、人が介入して、顧客による購入を抑止するビジネス・ルールをオーバーライドすることができます。ヒューマン・タスクには、属性がある場合があります (タスクの所有者を設定する属性や、指定された人が不在の場合に備えたエスカレーション・プロセスを提供する属性など)。ヒューマン・タスク・コンポーネントは、多くのプロセスで、検討、調査、承認などのタスクに人の介入が必要となることを認識しています。

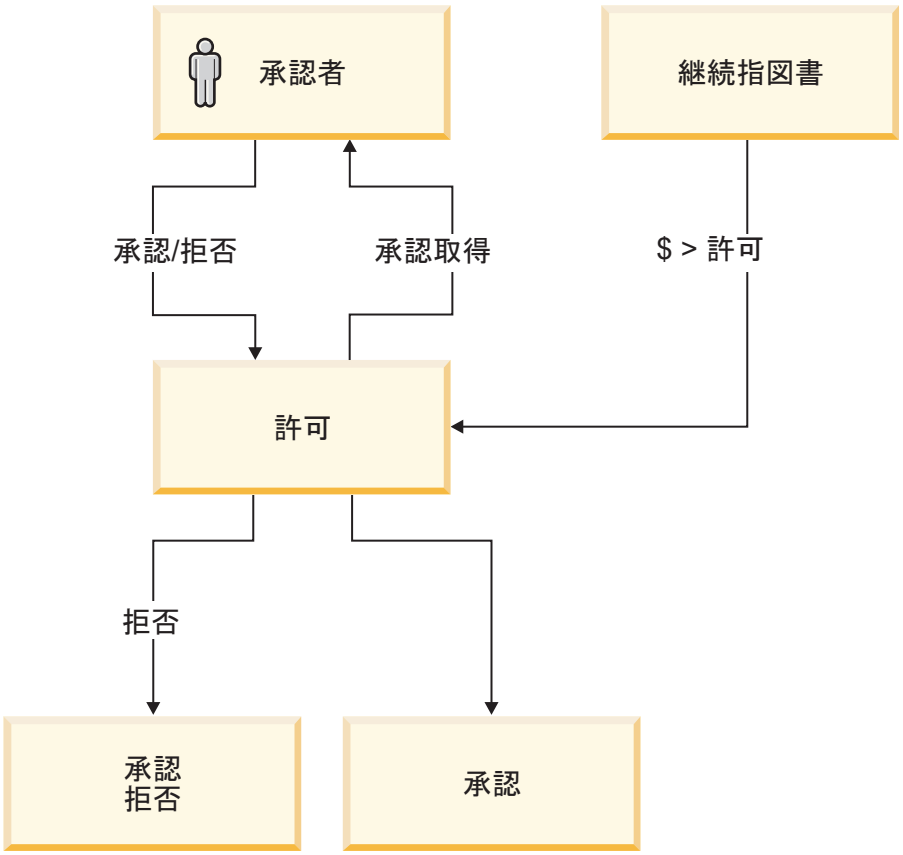


図 11. ヒューマン・タスク・コンポーネント

ヒューマン・タスクは、メディエーション・モジュールでは使用できません。WebSphere Process Server のみにデプロイできます。

インターフェース・マップ

インターフェース・マップ は、対話するコンポーネントのインターフェース間の違いを解決します。

注: インターフェース・マップは、WebSphere Process Server バージョン 7.0 以降では推奨されていません。WebSphere Integration Developer 内の既存のインターフェース・マップ・コンポーネントを、メディエーション・フロー・コンポーネント内の機能を使用するようにマイグレーションすることができます。

互いに対話する必要のあるコンポーネントのインターフェースが異なっていることがよくあります。WebSphere Integration Developer で、異なるアプリケーション用に作成されたコンポーネントを組み合わせることが多いため、このような違いが発生します。それらを再利用してアプリケーションを作成することが、WebSphere Integration Developer の強みの 1 つです。再利用しなければ、類似したコンポーネントを再コーディングすることになります。ただし、通常何らかの調整が必要です。

例えば、2 つのコンポーネントで、基本的に同じアクションを実行するが getCredit と getCreditRating などのように異なる名前を持つメソッドを持っていることがあります。操作名が異なる場合もあり、それらの操作のパラメーター・タイプが異なる

場合もあります。インターフェース・マップは、違いを解決して 2 つのコンポーネントが対話できるように、このようなメソッドの操作とパラメーターをマップします。インターフェース・マップは、2 つのコンポーネント間のブリッジのようなもので、違いがあってもそれらを相互にワイヤリングできるようにします。

インターフェース・マップは、それを使用するコンポーネントからは独立して存在しています。すなわち、コンポーネント自体を変更する必要はありません。

インターフェース・マップは、メディエーション・モジュールでは使用できません。WebSphere Process Server のみにデプロイできます。

メディエーション・フロー

メディエーション は、サービス間で動的に仲介または調停する方法です。メディエーション・フロー がメディエーションを実装します。

メディエーションには、いくつかの有用な機能があります。例えば、あるサービスからのデータを、後続のサービスで受け入れ可能な形式に変換する必要がある場合に、メディエーションを使用できます。サービスからのメッセージが次のサービスに送信される前に、ロギングにより、それらのメッセージをログに記録できます。ルーティングでは、あるサービスからのデータを、メディエーション・フローが決定した適切なサービスへ送付できます。メディエーションは、接続先のサービスからは独立して機能します。アセンブリー・エディター内のメディエーションは、エクスポートとインポート間のメディエーション・フロー・コンポーネントとして表示されます。

以下の図では、3 つのサービス要求元 (エクスポート) が、出力データをメディエーション・フロー・コンポーネントのインターフェースに送信します。メディエーション・フロー・コンポーネントは、次に、適切なデータを 2 つのサービス・プロバイダー (インポート) に送付します。

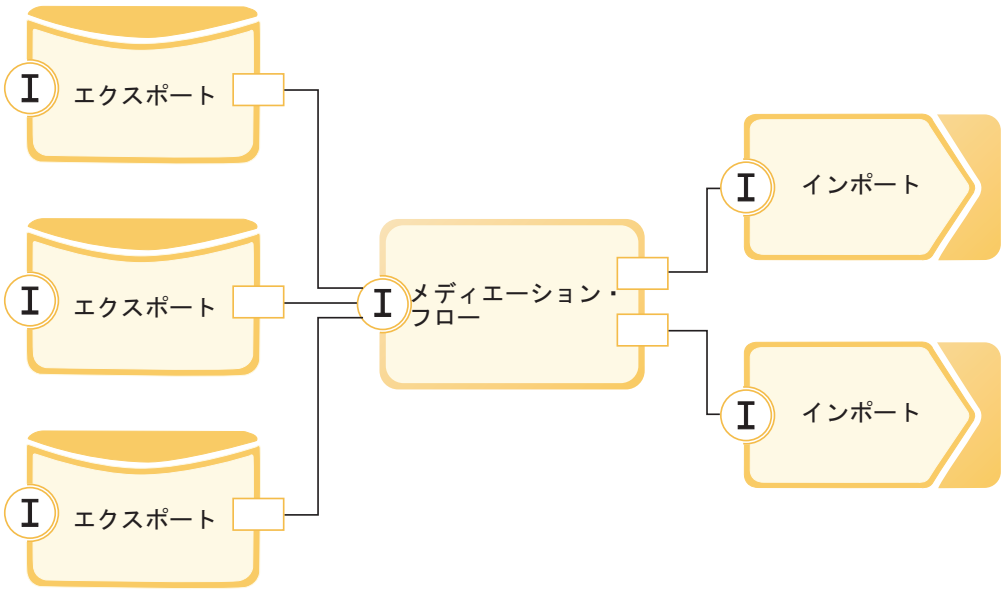


図 12. 3 つのサービス要求元 (エクスポート) および 2 つのサービス・プロバイダー (インポート) の間のメディエーション・フロー・コンポーネント

メディエーション・フローは、メディエーション・フロー・エディターで作成された、フロー状の構成体です。アセンブリー・エディターでメディエーション・フロー・コンポーネントを選択すると、メディエーション・フロー・エディターが起動されます。メディエーション・フロー・エディターでは、あるサービス (サービス要求元またはエクスポート) からの操作が、メディエーション・フロー・エディターが提供する機能と共に、別のサービス (サービス・プロバイダーまたはインポート) の操作にマップされます。メディエーション・フロー・エディターが提供する機能は、メディエーション・プリミティブ と呼ばれ、次の図に示すように、メディエーション・フローでワイヤリングされています。メディエーション・プリミティブは、IBM 提供のものがありますが、独自のカスタム・プリミティブを作成することもできます。メディエーション・プリミティブは、メッセージ内容とメッセージ・コンテキストの両方に作用することができます。コンテキストとは、SOAP または JMS ヘッダーや、ユーザー定義プロパティーなどの、バインディング固有の情報です。

次の図では、applyforLoan という操作が、まず、メッセージを記録する Log というロギング・プリミティブへメッセージを送信します。Log はメッセージを Filter プリミティブに送信します。Filter プリミティブは、メッセージに応じて、processBusinessLoan 操作または processPersonalLoan 操作のいずれかにメッセージを送付します。

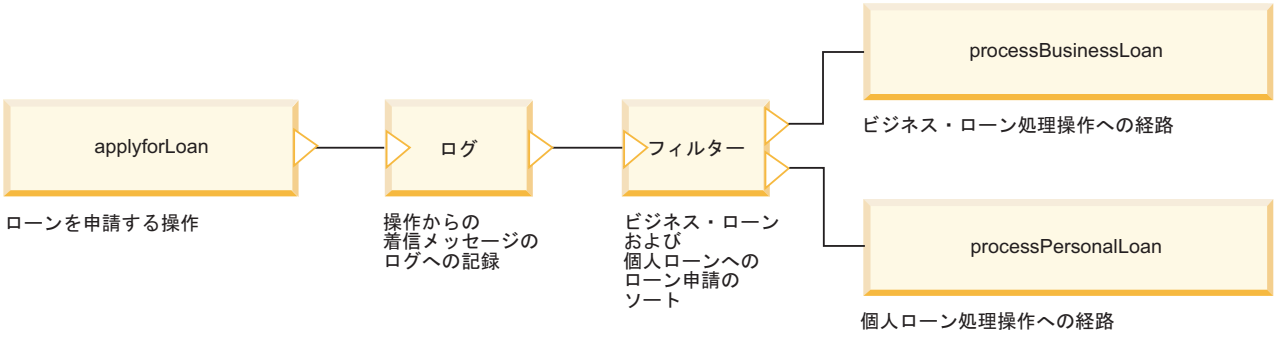


図 13. 操作間のメディエーション・フロー

モジュールセクションで説明するように、メディエーション・フローは、モジュールまたはメディエーション・モジュール内に存在できます。どちらのタイプのモジュールにも、1 つ以上のメディエーション・フロー・コンポーネントと、メディエーション・フロー・コンポーネントを拡張する 0 個以上の Java コンポーネントを含めることができます。モジュールは、WebSphere Process Server にデプロイできます。メディエーション・モジュールは、WebSphere Process Server または WebSphere Enterprise Service Bus サーバーのいずれかにデプロイすることができます。

サービス修飾子

アプリケーションは、*サービス修飾子* を指定することにより、ランタイム環境へのサービス品質 (QoS) ニーズと通信します。修飾子は、サービス・クライアントとターゲット・サービスとの間の相互作用を管理します。

修飾子はサービス・コンポーネント参照、インターフェース、および実装に指定することが可能です。QoS 値の宣言は実装の外部にあるため、実装を変更せずにこれらの値を変更したり、同じ実装の複数のインスタンスがさまざまなコンテキストで使用される場合に値がそれぞれ異なるように設定したりできます。

修飾子のカテゴリーは以下のとおりです。

- トランザクション - トランザクションのタイプに対するルール
- アクティビティー・セッション - アクティブ・セッションの結合に対するルール
- セキュリティー - アクセス権に対するルール
- 非同期信頼性 - 非同期メッセージ配信に対するルール

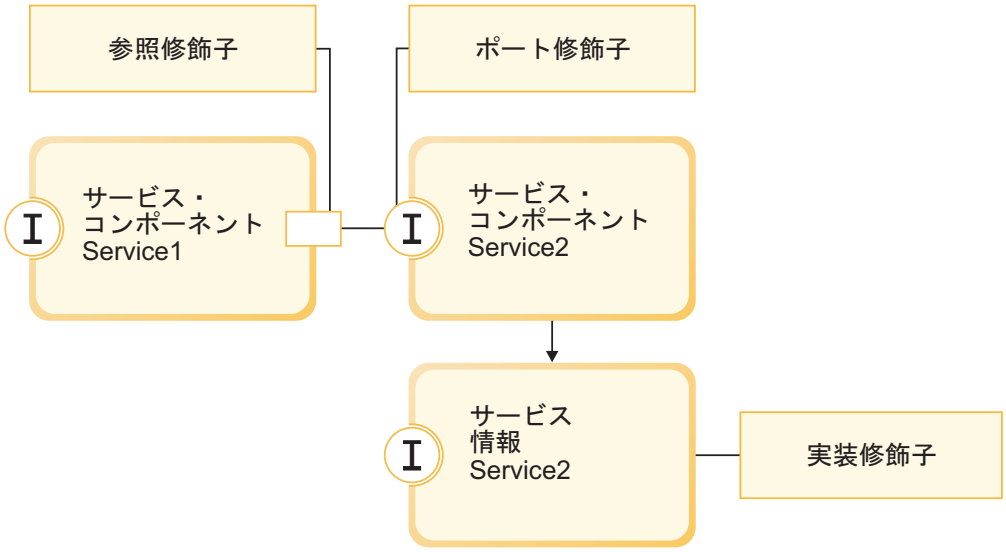


図 14. 修飾子

スタンドアロン参照

スタンドアロン参照 は、Service Component Architecture コンポーネントとして定義されていないアプリケーション (例えば JavaServer Pages やサーブレットなど) への参照です。スタンドアロン参照により、これらのアプリケーションは Service Component Architecture コンポーネントと対話できるようになります。

スタンドアロン参照には、インターフェースも実装もありません (実装はモジュールのスコープ外であるため)。モジュールには、スタンドアロン参照が含まれないか、1 つのスタンドアロン参照成果物が含まれるかのいずれかです。スタンドアロン参照には、WebSphere Integration Developer で作成された Service Component Architecture コンポーネントと共に既存のアプリケーションを使用できるという実用的な価値があります。

スタンドアロン参照は、メディエーション・モジュールで使用できます。これらは WebSphere Process Server または WebSphere Enterprise Service Bus サーバーのいずれかにデプロイできます。

ビジネス・オブジェクト

ビジネス・オブジェクトは、Service Component Architecture を補完するものです。Service Component Architecture は、サービスをコンポーネントとして定義し、それらの間の接続を定義します。ビジネス・オブジェクト は、コンポーネント間に流れるデータを定義します。

各コンポーネントは、情報を入力および出力として受け渡します。サービスが呼び出されると、データ・オブジェクトは、WSDL ポート・タイプの使用時には文書リテラル・エンコード方式によって XML 文書として渡され、Java インターフェースの使用時には Java オブジェクトとして渡されます。Service Component Architecture サービス内のデータおよびメタデータについては、データ・オブジェクトの形式が推奨されます。コンポーネントと同様に、ビジネス・オブジェクトは、データ・オ

ブジェクトを実装から分離しています。例えば、コンポーネントが購入注文を操作する場合に、購入注文自体は、JDBC、EJB、などを使用してデータへの更新を行っている可能性があります。ビジネス・オブジェクトは、統合開発者が、ビジネス成果物の処理に集中できるようにします。実際、統合開発者にとって、サービス・データ・オブジェクトは透過的です。それらは、サービス・データ・オブジェクト Java Specification Request (JSR) によって定義されます。

図 15 では、ビジネス・オブジェクトが外部サービスからエクスポートに、エクスポートからコンポーネントに、コンポーネントからコンポーネントに、コンポーネントからインポートに、そしてインポートからサービスに渡されます。インポートとエクスポートについては、 バインディングのセクションで説明します。

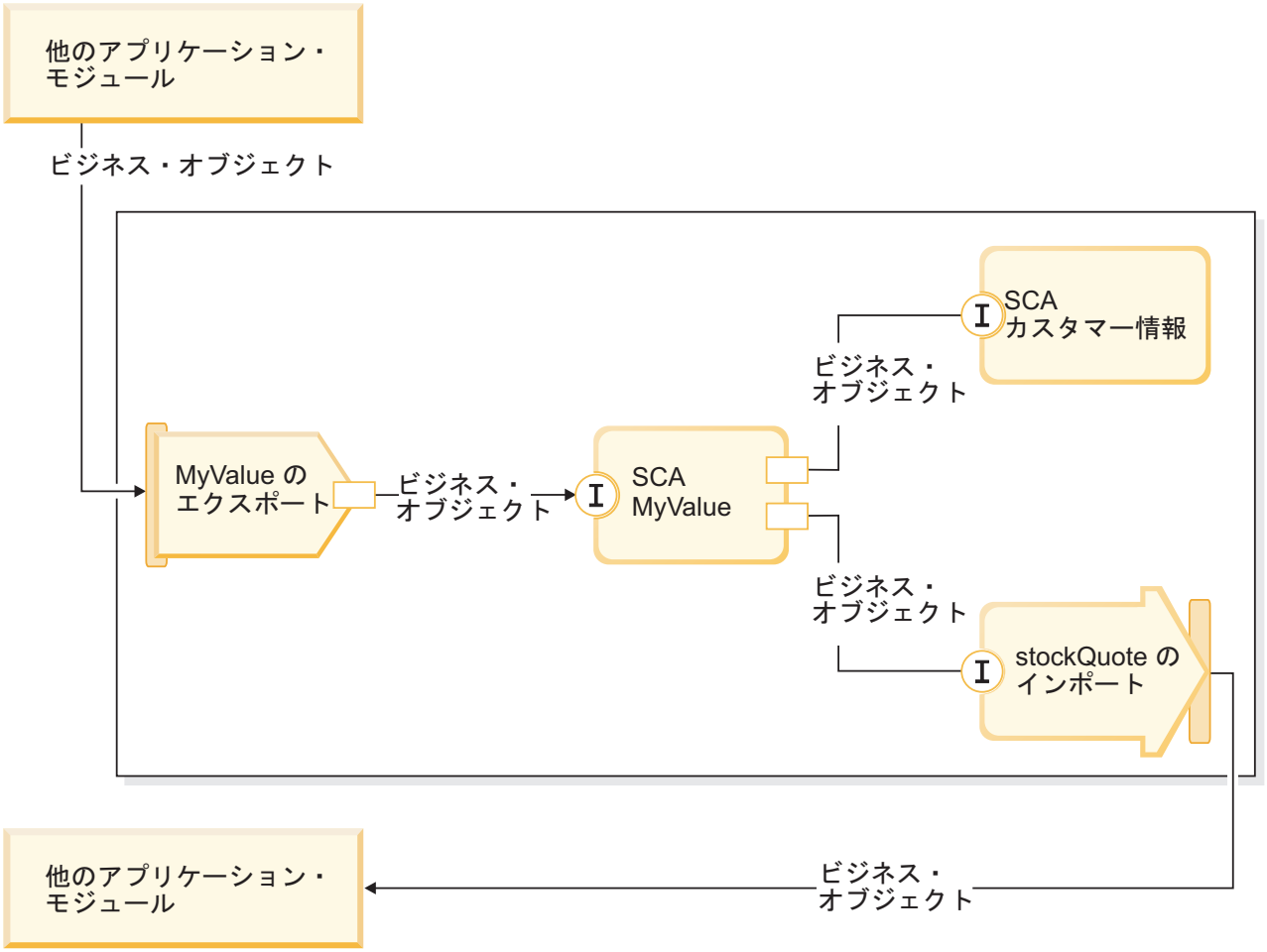


図 15. ビジネス・オブジェクト

ビジネス・オブジェクト

コンピューター・ソフトウェア業界では、ビジネス・オブジェクトを使用して、アプリケーション処理の対象となるビジネス・データを自然に表現するプログラミング・モデルとフレームワークをいくつか開発してきました。

一般に、これらのビジネス・オブジェクトには以下のような特徴があります。

- 業界標準を使用して定義される
- データベース表またはエンタープライズ情報システムにデータを透過的にマップする
- リモート呼び出しプロトコルをサポートする
- アプリケーション・プログラミング用のデータ・プログラミング・モデルの基盤を提供する

ツールの観点からは、WebSphere Integration Developer が、異なるドメインからのさまざまなビジネス・エンティティーを表すための共通ビジネス・オブジェクト・モデルを開発者に提供します。開発者が開発時にこのモデルを使用すると、ビジネス・オブジェクトを XML スキーマ定義として定義できるようになります。

XML スキーマ定義で定義されたビジネス・データは、実行時に Java ビジネス・オブジェクトとして表現されます。このモデルにおけるビジネス・オブジェクトは、初期版のサービス・データ・オブジェクト (SDO) 仕様に大まかに基づいており、ビジネス・データの操作に必要なプログラミング・モデル・アプリケーション・インターフェース一式が用意されています。

ビジネス・オブジェクトの定義

ビジネス・オブジェクトの定義は、WebSphere Integration Developer でビジネス・オブジェクト・エディターを使用して行います。ビジネス・オブジェクト・エディターはビジネス・オブジェクトを XML スキーマ定義として保管します。

XML スキーマを使用したビジネス・オブジェクトの定義には、次のようにいくつかの利点があります。

- XML スキーマは、標準ベースのデータ定義モデルと、異機種のシステム間とアプリケーション間の相互運用を実現するための基盤を提供します。XML スキーマは Web サービス記述言語 (WSDL) と連携して使用され、コンポーネント間、アプリケーション間、システム間の、標準ベースのインターフェース規約を提供します。
- XML スキーマは、リッチ・データ定義モデルを使用してビジネス・データを表現します。このモデルには、複合タイプ、単純タイプ、ユーザー定義タイプ、タイプの継承、カーディナリティーなどのフィーチャーが含まれています。
- ビジネス・オブジェクトは、ビジネス・インターフェースと、Web サービス記述言語で定義されたデータを使用して定義することができます。また、業界標準組織の XML スキーマや別のシステムまたはアプリケーションの XML スキーマを使用して定義することもできます。WebSphere Integration Developer では、こうしたビジネス・オブジェクトを直接インポートすることができます。

WebSphere Integration Developer は、データベースとエンタープライズ情報システムからビジネス・データをディスカバーし、このビジネス・データの標準ベース XML

スキーマのビジネス・オブジェクト定義を生成するための機能もサポートしています。この方法で生成されたビジネス・オブジェクトは、*アプリケーション固有のビジネス・オブジェクト*と呼ばれることがよくあります。これは、企業の情報システムで定義されたビジネス・データの構造と同じような構造がこのビジネス・オブジェクトで使用されるためです。

さまざまな情報システムのデータをプロセスで操作する場合は、ビジネス・データのさまざまな異なる表現 (CustomerEIS1 および CustomerEIS2、または OrderEIS1 および OrderEIS2 など) を 1 つの正規表現 (Customer または Order など) に変換することが重要になる場合があります。このような正規表現は、*汎用ビジネス・オブジェクト*と呼ばれることがよくあります。

ビジネス・オブジェクト定義は、多くの場合 (特に、汎用ビジネス・オブジェクトの場合)、複数のアプリケーションで使用されます。こうしたビジネス・オブジェクト定義の再使用をサポートするために、WebSphere Integration Developer では、ビジネス・オブジェクトをライブラリー内で作成して複数のアプリケーション・モジュールに関連付けることができます。

Service Component Architecture (SCA) アプリケーション・モジュールによって提供されて使用されるサービスの規約と、アプリケーション・モジュール内のコンポーネントの作成に使用される規約は、Web サービス記述言語 (WSDL) を使用して定義されます。 WSDL により、規約の操作とビジネス・オブジェクトの両方を記述することができます。これらの操作とビジネス・オブジェクトは、ビジネス・データを表現するために XML スキーマによって定義されます。

ビジネス・オブジェクトでの作業

Service Component Architecture (SCA) には、アプリケーション・モジュールを定義するためのフレームワーク、フレームワークが提供するサービス、フレームワークがコンシュームするサービス、アプリケーション・モジュールのビジネス・ロジックを提供するコンポーネント構成が用意されています。ビジネス・オブジェクトには、アプリケーション内で重要な役割があります。ビジネス・オブジェクトにより、サービス規約とコンポーネント規約を記述するためのビジネス・データと、コンポーネントで操作するビジネス・データが定義されます。

次の図は、SCA アプリケーション・モジュールを示したものです。この図では、さまざまな場所で開発者がビジネス・オブジェクトを使用して作業を行っています。

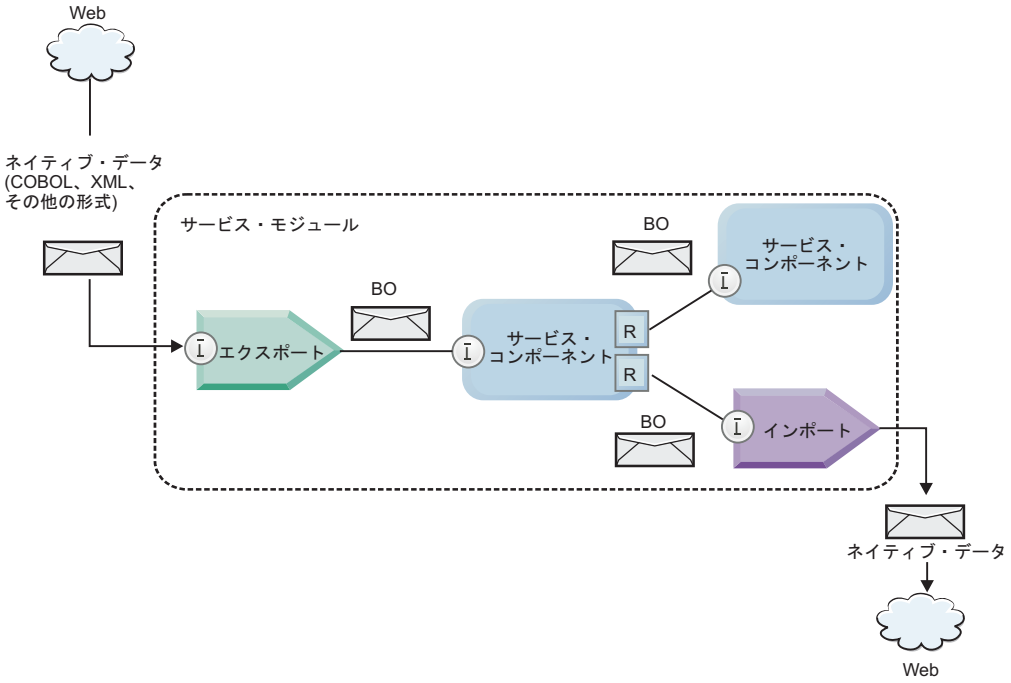


図 16. ビジネス・オブジェクトは、アプリケーション内のサービス間を流れるデータを表します。

注: このトピックでは、ビジネス・オブジェクトが SCA アプリケーション・モジュールでどのように使用されるかを説明します。Java インターフェースを使用する場合は、Java オブジェクトも SCA アプリケーション・モジュールで処理できます。

ビジネス・オブジェクト・プログラミング・モデル

ビジネス・オブジェクトのプログラミング・モデルは、一連の Java インターフェースで構成され、それらのインターフェースは次のものを表します。

- ビジネス・オブジェクト定義およびインスタンス・データ
- ビジネス・オブジェクトに対する操作をサポートする一連のサービス

ビジネス・オブジェクト・タイプ定義は、commonj.sdo.Type インターフェースおよび commonj.sdo.Property インターフェースで表されます、ビジネス・オブジェクト・プログラミング・モデルには、XML スキーマの複合タイプ情報を Type インターフェースにマップし、複合タイプ定義内の各エレメントを Property インターフェースにマップするための一連のルールが用意されています。

ビジネス・オブジェクト・インスタンスは commonj.sdo.DataObject interface で表されます。ビジネス・オブジェクト・プログラミング・モデルには、タイプは指定されていません。そのため、同じ commonj.sdo.DataObject インターフェースを使用して、Customer や Order などの異なるビジネス・オブジェクト定義を表すことができます。設定可能なプロパティーと、各ビジネス・オブジェクトから取得できるプロパティーについては、各ビジネス・オブジェクトに関連する XML スキーマ内のタイプ情報によって定義されます。

ビジネス・オブジェクト・プログラミング・モデルの動作は、サービス・データ・オブジェクト 2.1 仕様に基づきます。詳しくは、次の Web ページで SDO 2.1 for Java の仕様、チュートリアル、および Javadoc を参照してください。
http://osoa.org/display/Main/Service+Data+Objects+Specifications

ビジネス・オブジェクト・サービスは、ビジネス・オブジェクト上のさまざまなライフサイクル操作 (作成、同等化、構文解析、直列化など) をサポートしています。

ビジネス・オブジェクト・プログラミング・モデルの詳細については、ビジネス・オブジェクト・サービスを使用したプログラミング および Package com.ibm.websphere.bo を参照してください。

バインディング、データ・バインディング、およびデータ・ハンドラー

23 ページの図 16 に示すように、SCA アプリケーション・モジュールが提供するサービスの呼び出しに使用されるビジネス・データは、ビジネス・オブジェクトに変換されます。これにより、SCA コンポーネントでビジネス・データを操作できるようになります。同様に、SCA コンポーネントによって操作されるビジネス・オブジェクトは、外部サービスで必要なデータ・フォーマットに変換されます。

サービスのエクスポートとインポートで使用されるバインディングにより、データ・フォーマットが自動的に正しい形式に変換される場合があります (Web サービス・バインディングなど)。また、非ネイティブの形式を DataObject インターフェースによって表されるビジネス・オブジェクトに変換するためのデータ・バインディングまたはデータ・ハンドラーを開発者側で準備できる場合もあります (JMS バインディングなど)。

データ・バインディングおよびデータ・ハンドラーの開発について詳しくは、データ・ハンドラー および データ・バインディング を参照してください。

コンポーネント

SCA コンポーネントは、Web サービス記述言語と XML スキーマを組み合わせて使用することにより、サービスの提供とコンシュームに関する規約を定義します。SCA がコンポーネント間で受け渡しをするビジネス・データは、DataObject インターフェースを使用してビジネス・オブジェクトとして表されます。 SCA は、これらのビジネス・オブジェクトのタイプが、呼び出し対象のコンポーネントによって定義されたインターフェース規約と互換性があるかどうかを検証します。

ビジネス・オブジェクトを操作するためのプログラミング・モデル抽象化は、コンポーネントによって異なります。POJO コンポーネントとメディエーション・フロー・コンポーネントの Custom プリミティブは、ビジネス・オブジェクト・プログラミングのインターフェースとサービスを直接使用して Java プログラミングを使用可能にすることにより、ビジネス・オブジェクトの直接操作を可能にしています。多くのコンポーネントで、ビジネス・オブジェクトを操作するための高水準の抽象化が用意されていますが、ビジネス・オブジェクトのインターフェースとサービスにおけるカスタムの動作を定義するための Java コードのスニペットも用意されています。

Interface Flow Mediation と Business Object Map コンポーネントの組み合わせ、またはメディエーション・フロー・コンポーネントとその XML マップ・プリミティブの組み合わせを使用して、ビジネス・オブジェクトを変換することができます。これらのビジネス・オブジェクト変換機能は、アプリケーション固有のビジネス・オブジェクトを汎用的なビジネス・オブジェクトに変換 (あるいはその逆) する場合に便利です。

特殊なビジネス・オブジェクト

サービス・メッセージ・オブジェクトとビジネス・グラフは、特定のアプリケーションの目的に使用される 2 種類の専門化されたビジネス・オブジェクトです。

サービス・メッセージ・オブジェクト

サービス・メッセージ・オブジェクト (SMO) は、サービス呼び出しに関連するデータ・コレクションを表すためにメディエーション・フロー・コンポーネントによって使用される特殊なビジネス・オブジェクトです。

SMO には、固定された最上位階層があります。この階層は、ヘッダー、コンテキスト、本文、添付ファイル (存在する場合) から構成されています。

- ヘッダーは、特定のプロトコルまたはバインディングを使用して、サービス呼び出しに関連する情報を渡します。例として、SOAP ヘッダーや JMS ヘッダーがあります。
- コンテキスト・データは、メディエーション・フロー・コンポーネントによる処理中に、呼び出しに関する追加の論理情報を渡します。通常、この情報は、クライアントによって送受信されるアプリケーション・データの一部ではありません。
- SMO の本文は、ペイロード・ビジネス・データを渡します。このデータは、標準的なビジネス・オブジェクトの形式で、コア・アプリケーション・メッセージまたは呼び出しデータを表現します。

SMO は、添付ファイルが設定された SOAP を使用して、Web サービス呼び出し用の添付データを渡すこともできます。

メディエーション・フローは、要求のルーティングやデータ変換などのタスクを実行します。SMO は、ヘッダーとペイロードの内容を 1 つの統一された構造で組み合わせて表現します。

ビジネス・グラフ

ビジネス・グラフは、統合のシナリオにおけるデータ同期をサポートする場合に使用される特殊なビジネス・オブジェクトです。

ここでは、特定の発注データを表示する 2 つのエンタープライズ情報システムを例として考えてみます。一方のシステムで発注データが変更されると、発注データを同期化するためのメッセージがもう一方のシステムに送信されます。ビジネス・グラフは、注文データの変更された部分だけを他方のシステムに送信し、変更のタイプを定義するための変更概要情報を注釈として発注データに追加するという概念をサポートしています。

この例では、発注データの明細行項目の 1 つが削除され、予想出荷日プロパティーが更新されていることが、発注ビジネス・グラフから他方のシステムに通知されます。

ビジネス・グラフは、WebSphere Integration Developer の既存のビジネス・オブジェクトに簡単に追加することができます。ビジネス・グラフが最も頻繁に使用されるのは、WebSphere アダプターを使用していて、WebSphere InterChange Server アプリケーションのマイグレーションをサポートする場合のシナリオです。

ビジネス・オブジェクト構文解析モード

WebSphere Integration Developer には、モジュールとライブラリーのプロパティーが用意されています。このプロパティーを使用すると、ビジネス・オブジェクト用の XML 構文解析モード (EAGER または LAZY) を構成することができます。

- このオプションを「*EAGER*」に設定すると、XML バイト・ストリームの構文解析が優先的に実行され、ビジネス・オブジェクトが作成されます。
- このオプションを「*LAZY*」に設定すると、ビジネス・オブジェクトは通常どおりに作成されますが、XML バイト・ストリームの実際の構文解析は、ビジネス・オブジェクト・プロパティーにアクセスしたときにのみ、部分的または遅延して実行されます。

どちらの XML 構文解析モードでも、非 XML データの構文解析は常に優先的に実行され、ビジネス・オブジェクトが作成されます。

LAZY 構文解析モードと EAGER 構文解析モードを使用する場合のそれぞれの利点

XML 構文解析を行う場合、LAZY 構文解析モードが適しているアプリケーションもあれば、EAGER 構文解析モードによってパフォーマンスが向上するアプリケーションもあります。両方の構文解析モードでベンチマーク評価を行い、アプリケーションの特定の特性に最も適したモードを判断することをお勧めします。

このセクションには、それぞれの構文解析モードに適したアプリケーションのタイプに関する一般的なガイダンスがあります。

- LAZY 構文解析モードが適したアプリケーション

  大きな XML データ・ストリームを構文解析するアプリケーションの場合、LAZY 構文解析モードを使用するとパフォーマンスが向上する可能性があります。XML バイト・ストリームのサイズが大きくなり、アプリケーションがアクセスするバイト・ストリームからのデータ量が小さくなるにつれて、パフォーマンス上の利点が大きくなります。

  注: ビジネス・オブジェクトの LAZY 構文解析モードは、WebSphere Process Server 7.0.0.3 以降のバージョンでサポートされます。メディエーション・フロー・コンポーネントを含むモジュールおよびメディエーション・モジュールはサポートされません。
- EAGER 構文解析モードが適したアプリケーション

  次のようなアプリケーションでは、EAGER 構文解析モードによりパフォーマンスが向上する可能性があります。

– 非 XML データ・ストリームを構文解析するアプリケーション
– BOFactory サービスを使用した作成されたアプリケーション
– 非常に小さい XML メッセージを構文解析するアプリケーション

アプリケーションのマイグレーションと開発に関する考慮事項

元々 EAGER 構文解析モードを使用して開発されたアプリケーションを、LAZY 構文解析モードを使用するように構成する場合、あるいはアプリケーションを EAGER 構文解析モードと LAZY 構文解析モードの間で切り替える計画の場合は、これらのモードの相違と、モードを切り替える際の考慮事項を知っている必要があります。

エラー処理

構文解析する XML バイト・ストリームの形式が正しくない場合、構文解析例外が発生します。

- EAGER XML 構文解析モードでは、ビジネス・オブジェクトがインバウンド XML ストリームから構文解析されるとすぐにこれらの例外が発生します。
- LAZY 構文解析モードが構成されている場合、ビジネス・オブジェクト・プロパティーがアクセスされて形式の正しくない XML が構文解析されると、構文解析例外が潜在的に発生します。

形式の正しくない XML を処理するには、次のいずれかのオプションを選択します。

- エッジにエンタープライズ・サービス・バスをデプロイしてインバウンド XML を検証する。
- ビジネス・オブジェクト・プロパティーがアクセスされる点に LAZY エラー検出ロジックを作成する。

例外スタックとメッセージ

EAGER 構文解析モードと LAZY 構文解析モードでは基礎となる実装が異なるため、ビジネス・オブジェクト・プログラミング・インターフェースおよびサービスによってスローされたスタック・トレースが同じ例外クラス名を持っていても、例外メッセージや、実装環境固有の例外クラスのラップ・セットは同じであるとは限りません。

XML 直列化形式

LAZY 構文解析モードでは、直列化が行われると、インバウンド・バイト・ストリームからアウトバウンド・バイト・ストリームに未変更 XML をコピーしようとするパフォーマンス最適化が提供されます。その結果、パフォーマンスは向上します。ただし、ビジネス・オブジェクト全体が LAZY 構文解析モードで更新された場合や、EAGER 構文解析モードで実行されていた場合は、アウトバウンド XML バイト・ストリームの直列化形式が異なるものになる可能性があります。

XML 直列化形式は構文としては正確に同じでない場合がありますが、ビジネス・オブジェクトで提供される意味値は構文解析モードに関わらず等価です。そのため、アプリケーションが異なる構文解析モードで実行されていても意味的に等価であれば、それらのアプリケーション間で XML を安全に受け渡すことができます。

ビジネス・オブジェクト・インスタンス・バリデーター

LAZY XML 構文解析ビジネス・オブジェクト・モードのインスタンス・バリデーターを使用すると、ビジネス・オブジェクトの検証 (特にプロパティー値のファセット検証) の精度が高くなります。この改善により、LAZY 構文解析モードのインスタンス・バリデーターでは、EAGER 構文解析モードではキャッチできない追加の問題が検出され、より詳細なエラー・メッセージが出力されます。

バージョン 602 の XML マップ

元々 WebSphere Integration Developer バージョン 6.1 より前に開発されたメディエーション・フローには、LAZY 構文解析モードでは直接に処理できないマップまたはスタイル・シートを使用する XSLT プリミティブが含まれている場合があります。LAZY 構文解析モードで使用するためにアプリケーションをマイグレーションすると、XSLT プリミティブに関連付けられたマップ・ファイルがマイグレーション・ウィザードで自動的に更新されて、新しいモードで実行できるようになります。ただし、手動で編集されたスタイル・シートを XSLT プリミティブが直接参照している場合、このスタイル・シートはマイグレーションされず、LAZY XML 構文解析モードで実行することはできません。

非公開のプライベート API

非公開でプライベート、かつ実装固有のビジネス・オブジェクト・プログラミング・インターフェースを利用しているアプリケーションの場合、構文解析モードを切り替えるとコンパイルが失敗する可能性があります。EAGER 構文解析モードでは、こうしたプライベート・インターフェースは、通常、Eclipse Modeling Framework (EMF) によって定義されるビジネス・オブジェクトの実装クラスです。

すべての場合において、プライベート API をアプリケーションから削除することをお勧めします。

サービス・メッセージ・オブジェクト EMF API

WebSphere Process Server のメディエーション・コンポーネントには、com.ibm.websphere.sibx.smobo パッケージ提供されている Java のクラスとインターフェースを使用してメッセージの内容を操作するための機能が用意されています。LAZY XML 構文解析モードでは、com.ibm.websphere.sibx.smobo パッケージの Java インターフェースを引き続き使用することができますが、Eclipse Modeling Framework (EMF) のクラスとインターフェースを直接参照しているメソッドや EMF インターフェースから継承されたメソッドは失敗する可能性があります。

ServiceMessageObject とその内容を、LAZY XML 構文解析モードで EMF オブジェクトにキャストすることはできません。

BOMode サービス

BOMode サービスは、現在処理中の XML 構文解析が EAGER モードと LAZY モードのどちらで実行されているかを確認するために使用します。

マイグレーション

バージョン 7.0.0.0 より前のすべてのアプリケーションは EAGER 構文解析モードで実行されます。このアプリケーションが BPM ランタイム・マイグレーション・ツールを使用してマイグレーションされたランタイムである場合も、引き続き EAGER XML 構文解析モードで実行されます。

バージョン 7.0.0.0 より前のアプリケーションで LAZY XML 構文解析モードを使用できるように構成するには、最初に WebSphere Integration Developer を使用して、そのアプリケーションの成果物をマイグレーションする必要があります。マイグレーションが終了したら、LAZY XML 構文解析モードを使用するようにアプリケーションを構成します。

WebSphere Integration Developer の成果物のマイグレーションについては、『ソース成果物のマイグレーション (Migrating source artifacts)』を参照してください。解析モードの設定については、『モジュールとライブラリーのビジネス・オブジェクト解析モードの構成 (Configuring the business object parsing mode of modules and libraries)』を参照してください。

ビジネス・オブジェクト・プロパティー・タイプ QName

EAGER 構文解析を使用していたアプリケーションが LAZY 構文解析を使用するようにするには、タイプ QName のプロパティーを含むビジネス・オブジェクトを処理するように、アプリケーション・コードを変更する必要があります。EAGER 構文解析モードでは、WebSphere Process Server は Java クラス `org.eclipse.emf.ecore.xml.type.internal.QName` を使用して QName タイプのプロパティー値を定義します。LAZY 構文解析モードでは、Java クラス `javax.xml.namespace.QName` を使用して QName タイプ・プロパティーの値を設定します。モジュールのモードを EAGER から LAZY に変更するときは、Java クラス `org.eclipse.emf.ecore.xml.type.internal.QName` への参照を `javax.xml.namespace.QName` で置き換えて、アプリケーション・コードを変更してください。

リレーションシップ

リレーションシップとは、複数のデータ・エンティティー (通常はビジネス・オブジェクト) 間の関連です。リレーションシップを使用すると、ビジネス・オブジェクトやその他のデータ間で等価であるが、表現が異なるデータを変換したり、異なるアプリケーションで検出された異なるオブジェクトを関連で結ぶことができます。リレーションシップは、アプリケーション間やソリューション間で共有することができ、製品間であっても共有できます。

WebSphere Process Server のリレーションシップ・サービスでは、リレーションシップを管理するためのインフラストラクチャーおよび操作を提供します。ビジネス・オブジェクトがどこにあるかに関わらずユーザーがそれらを処理できるようになるため、リレーションシップ・サービスは、企業内のすべてのアプリケーションにわたる包括的なビューを提供し、BPM ソリューションのビルディング・ブロックとして機能することができます。リレーションシップは拡張可能であり、管理しやすいため、複雑な統合ソリューションで使用できます。

リレーションシップとは何か

リレーションシップは、ビジネス・オブジェクト間の関連です。リレーションシップ内の各ビジネス・オブジェクトは、リレーションシップの*参加者* と呼ばれます。リレーションシップ内の各参加者は、そのリレーションシップ内で参加者が果たす機能 (*ロール*) に基づいて他の参加者から区別されます。リレーションシップにはロールのリストが含まれています。

リレーションシップ「*定義*」は、各ロールについて記述し、ロールがどのように関係するかを指定します。また、リレーションシップの全体的な「形状」も記述します。例えば、あるロールに含まれるのは 1 人の参加者のみですが、別のあるロールには必要な数だけの参加者を含めることができます。一例として、*自動車* と*所有者* のリレーションシップを定義できます。このリレーションシップでは、1 人の所有者が複数の自動車を所有する場合があります。例えばある 1 つのインスタンスでは、これらのロールのそれぞれに以下の参加者を含めることができます。

- 自動車 (Ferrari)
- 所有者 (John)

リレーションシップ定義は、リレーションシップ・*インスタンス* のテンプレートです。インスタンスは、実行時にリレーションシップをインスタンス化したものです。上記の*自動車* と*所有者* の例では、インスタンスに以下のような関連をどれでも記述できます。

- John が Ferrari を所有する
- Sara が Mazda を所有する
- Bob が Ferrari を所有する

リレーションシップを使用すると、ビジネス・ロジック内でパーシスタンスを追跡するリレーションをカスタム・ビルドする必要がなくなります。ある特定のシナリオでは、リレーションシップ・サービスがユーザーに代わってすべての作業を行います。ID リレーションシップに関するセクションに記載された例を参照してください。

シナリオ

ここでは、統合ソリューションがリレーションシップを使用する可能性がある場合の標準的な例を示します。大企業が、複数の会社または事業単位を買収するものとします。各事業単位では、従業員とラップトップをモニターするためにそれぞれ異なるソフトウェアを使用しています。会社は従業員と従業員のラップトップをモニターする方法を必要としています。会社は、以下の操作が可能なソリューションを導入したいと考えます。

- さまざまな事業単位のすべての従業員を、1 つのデータベースに含まれているかのように表示する。
- 従業員のすべてのラップトップを単一のビューで表示する。
- 従業員がシステムにログオンしてラップトップを購入できるようにする。
- さまざまな事業単位で異なるエンタープライズ・アプリケーション・システムを許容する。

これを実現するには、例えば、異なるアプリケーション内の John Smith と John A. Smith が同じ従業員として認識されるようにする手段が会社で必要となります。例えば、複数のアプリケーション・スペースにまたがって単一のエンティティーを統合する手段が必要です。

さらに複雑なリレーションシップのシナリオでは、複数のアプリケーションで検出されたさまざまなオブジェクトをリレーションシップで結ぶビジネス・プロセスを作成します。複雑なリレーションシップのシナリオでは、ビジネス・オブジェクトはアプリケーション内ではなく統合ソリューション内にあります。リレーションシップ・サービスは、リレーションシップを永続的に管理するためのプラットフォームを提供します。リレーションシップ・サービスの前に、独自のオブジェクト・パーシスタンス・サービスを作成する必要があります。複雑なリレーションシップ・シナリオの 2 つの例を以下に示します。

- SAP アプリケーション内に VIN 番号を持つ car ビジネス・オブジェクトを保有しており、この自動車がだれか別のユーザーに所有されているという事実を追跡しようとしています。しかし、所有権リレーションシップは、PeopleSoft アプリケーション内のユーザーとつながっています。このパターンのリレーションシップでは、ソリューションが 2 つあり、それらの間をつなぐものを構築する必要があります。
- ある大規模な小売会社が、現金またはクレジットでの返金を求めて返品された商品をモニターするための機能を必要としています。これには 2 つの異なるアプリケーションが必要です。購入に対応する注文管理システム (OMS) と、返品に対応する返品管理システム (RMS) です。ビジネス・オブジェクトは複数のアプリケーションに存在しており、それらの間に存在するリレーションシップを確認する手段が必要です。

一般的な使用パターン

最も一般的なリレーションシップ・パターンは、*等価* パターンです。これらは相互参照、すなわち相関に基づいています。このパターンに適合する 2 つのタイプのリレーションシップがあります。*非 ID* および *ID* です。

- **非 ID リレーションシップ**は、ビジネス・オブジェクトまたはその他のデータ間に、1 対多または多対多ベースの関連を確立します。リレーションシップ・インスタンスごとに、各参加者の 1 つ以上のインスタンスが存在できます。非 ID リレーションシップの一種として、静的ルックアップ・リレーションシップがあります。SAP アプリケーション内の CA が Siebel アプリケーション内の California に関連しているリレーションシップが、その一例です。
-

  ID リレーションシップは、ビジネス・オブジェクトまたはその他のデータ間に、1 対 1 ベースの関連を確立します。リレーションシップ・インスタンスごとに、各参加者のインスタンスは 1 つのみ存在できます。ID リレーションシップは、意味的に等価であるが、異なるアプリケーション内では異なって識別されるビジネス・オブジェクト間の相互参照を取り込みます。リレーションシップ内の各参加者は、オブジェクトを一意的に識別する値 (または値の組み合わせ) を持つビジネス・オブジェクトに関連付けられています。 ID リレーションシップは通常、ID 番号や製品コードなどの、ビジネス・オブジェクトのキー属性を変換します。

例えば、SAP、PeopleSoft、および Siebel アプリケーションに car ビジネス・オブジェクトがあり、それらを同期化するソリューションを構築したい場合、通常は以下の 6 つのマップで手製のリレーションシップ同期化ロジックを導入する必要があります。

SAP -> 汎用

汎用 -> SAP

PeopleSoft -> 汎用

汎用 -> PeopleSoft

Siebel -> 汎用

汎用 -> Siebel

ただし、ソリューションでリレーションシップを使用する場合、リレーションシップ・サービスは、これらすべてのリレーションシップ・インスタンスをユーザーに代わって保守する事前作成パターン実装を提供します。

リレーションシップの操作用のツール

WebSphere Integration Developer の*リレーションシップ・エディター* は、ビジネス・インテグレーション・リレーションシップおよびロールをモデル化および設計するために使用するツールです。リレーションシップの作成およびリレーションシップ・エディターの使用に関する詳細な背景情報およびタスク情報については、WebSphere Integration Developer インフォメーション・センターを参照してください。

リレーションシップ・サービス は、WebSphere Process Server 内のインフラストラクチャー・サービスであり、システム内のリレーションシップおよびロールを保守し、リレーションシップおよびロールを管理するための操作を提供します。

リレーションシップ・マネージャー は、リレーションシップを管理するための管理用インターフェースです。管理コンソールの「リレーションシップ・マネージャー」ページを通じてアクセスします。

リレーションシップは、リレーションシップ・サービス API を通じてプログラマチックに呼び出すことができます。

リレーションシップ・サービス

リレーションシップ・サービスは、リレーションシップ・テーブル内にリレーションシップ・データを保管し、このテーブルで、複数のアプリケーションおよびソリューションにわたるアプリケーション固有の値を追跡します。リレーションシップ・サービスは、リレーションシップおよびロールの管理用の操作を提供します。

リレーションシップの機能の仕組み

リレーションシップおよびロールは、WebSphere Integration Developer 内のリレーションシップ・エディター・ツールのグラフィカル・インターフェースを使用して定義されます。リレーションシップ・サービスは、リレーションシップ・サービスの構成時に指定したデフォルト・データ・ソース内のリレーションシップ・データベースにあるテーブルに、相関データを格納します。リレーションシップ内の参加者ごとの情報が、別々のテーブル (参加者テーブルと呼ばれることもある) に保管され

ます。リレーションシップ・サービスは、これらのリレーションシップ・テーブルを使用して、関連するアプリケーション固有の値を追跡し、更新された情報をすべてのソリューションに伝搬させます。

リレーションシップはビジネス成果物であり、プロジェクト内部または共有ライブラリー内にデプロイされます。最初のデプロイメントで、リレーションシップ・サービスがデータを取り込みます。

実行時に、マップまたは WebSphere Process Server の他のコンポーネントでリレーションシップ・インスタンスが必要になると、リレーションシップのインスタンスが、シナリオに応じて更新または取得されます。

リレーションシップおよびロールのインスタンス・データは、以下の 3 つの方法で操作できます。

- WebSphere Process Server コンポーネント Java スニペットによるリレーションシップ・サービス API の呼び出し
- WebSphere Process Server ビジネス・オブジェクト・マッピング・サービスでのリレーションシップ変換
- リレーションシップ・マネージャー・ツール

リレーションシップの作成、リレーションシップ・タイプの識別、およびリレーションシップ・エディターの使用に関する背景情報およびタスク情報について詳しくは、WebSphere Integration Developer インフォメーション・センターを参照してください。

リレーションシップ・マネージャー

リレーションシップ・マネージャーは、リレーションシップを管理するための管理用インターフェースです。管理コンソールの「リレーションシップ・マネージャー」ページを通じてアクセスします。

リレーションシップ・マネージャーには、実行時にリレーションシップおよびロール・データを作成および操作するためのグラフィカル・ユーザー・インターフェースがあります。全レベルでリレーションシップ・エンティティーを管理できます。つまり、リレーションシップ・インスタンス、ロール・インスタンス、属性データ、プロパティー・データの各レベルです。リレーションシップ・マネージャーを使用して、以下のことを実行できます。

- システム内のリレーションシップのリストおよび個々のリレーションシップの詳細情報の表示
- リレーションシップ・インスタンスの管理:
  - インスタンス・データのサブセットを表示するためのリレーションシップ・データの照会
  - データベース・ビューを使用してインスタンス・データのサブセットを表示するためのリレーションシップ・データの照会
  - リレーションシップ照会に一致するリレーションシップ・インスタンスのリストとインスタンスに関する詳細情報の表示
  - リレーションシップ・インスタンスのプロパティー値の編集
  - リレーションシップ・インスタンスの作成および削除。

- ロールおよびロール・インスタンスの管理:
  - ロールまたはロール・インスタンスに関する詳細の表示
  - ロール・インスタンス・プロパティーの編集
  - リレーションシップのロール・インスタンスの作成および削除
  - データが信頼できることが分かっている時点までのリレーションシップ・インスタンス・データのロールバック
- 既存の静的リレーションシップからシステムへのデータのインポート、または既存の静的リレーションシップから RI ファイルまたは CSV ファイルへのデータのエクスポート
- リレーションシップを使用するアプリケーションのアンインストール時の、そのリレーションシップのスキーマおよびデータのリポジトリーからの削除

Network Deployment 環境のリレーションシップ

リレーションシップは、追加の構成を行わなくても Network Deployment (ND) 環境で使用できます。

Network Deployment (ND) 環境では、リレーションシップはアプリケーション・クラスターにインストールされます。これにより、リレーションシップはクラスター内で可視となり、クラスター内のすべてのサーバーが、リレーションシップ・データベースに格納されたインスタンス・データにアクセスできるようになります。ND 環境でリレーションシップ・サービスを実行することができると、リレーションシップ・サービスの拡張が容易になり、可用性が高くなります。

リレーションシップ・マネージャーを使用すると、集中管理インターフェースを通じて、さまざまなクラスターにまたがってリレーションシップを管理できます。リレーションシップ MBean を選択することによって、リレーションシップ・マネージャーをクラスター内のサーバーに接続します。

リレーションシップ・サービス API

リレーションシップは、ビジネス・オブジェクト・マップの内部または外部で、リレーションシップ・サービス API を通じてプログラマチックに呼び出すことができます。

以下の 3 つの API タイプが使用可能です。

- リレーションシップ・インスタンス操作 API (インスタンス・データを直接作成、更新、削除する操作を含む)
- リレーションシップ・パターン・サポート API (correlate() 、correlateforeignKeyLookup を含む)
- リレーションシップ・ルックアップ・パターン (ルックアップ API)

WebSphere Process Server のエンタープライズ・サービス・バス

WebSphere Process Server では、WebSphere Enterprise Service Bus と同じ機能を含む、アプリケーション・サービスの統合がサポートされます。

エンタープライズ・サービス・バスを介したサービスの接続

エンタープライズ・サービス・バス (ESB) を使用して、SOA の柔軟性を最大化することができます。サービス対話の参加者は、直接お互いに接続するのではなく、ESB に接続します。

サービス・リクエスターが ESB に接続すると、ESB は、必要な機能とサービスの品質を提供するサービス・プロバイダーにメッセージを使用してその要求を配信する責任を負います。ESB はリクエスターとプロバイダーの対話を容易にし、プロトコル、対話パターン、またはサービス機能のミスマッチに対処します。ESB はまた、モニターと管理を使用可能にしたり拡張したりすることができます。ESB は SOA の中核機能を実装し拡張する、仮想化および管理の機能を提供します。

ESB には、以下のような特徴的な機能があります。

ロケーションと ID
参加者は、その他の参加者のロケーションまたは ID を知る必要がありません。例えば、要求側は、いくつもある中のどのプロバイダーによって、要求サービスが実行されるのかを知る必要がありません。サービス・プロバイダーを追加または削除しても、混乱は生じません。

対話プロトコル
参加者は、同じ通信プロトコルまたは対話スタイルを共有する必要がありません。例えば、SOAP over HTTP として表現された要求は、 Java メッセージ・サービス (JMS) を介してのみ SOAP を理解するプロバイダーによってサービスを受けることができます。

インターフェース
要求側とプロバイダーは、共通インターフェースについて一致する必要がありません。 ESB は、要求メッセージと応答メッセージをプロバイダーが期待する形式に変換することによって、違いを調整します。

要求側とプロバイダーは、共通インターフェースについて一致する必要がありません
ESB は、要求メッセージをプロバイダーが期待する形式に変換することによって、違いを調整します。

(対話) サービスの品質
参加者 (または、システム管理者) は、サービス品質要件を宣言します。これには、要求の許可、メッセージ内容の暗号化と暗号化解除、サービス対話の自動監査、要求の送付方法 (速度またはコストの最適化など) が含まれます。

参加者の間に ESB を介在させることによって、メディエーション と呼ばれる論理構成を介して、対話を調整することが可能になります。メディエーションは、要求側とプロバイダーの間でやり取りされるメッセージに対して機能します。例えば、メディエーションを使用して、要求側が求める固有の特性を備えたサービスを検出したり、要求側とプロバイダーの間のインターフェースの相違を解決したりすることができます。複雑な対話の場合は、メディエーションを連続的に続けることもできます。

エンタープライズ・サービス・バスは、メディエーションを使用して、要求側とサービスの間で以下のアクションを実行します。

- サービス間のメッセージのルーティング。エンタープライズ・サービス・バスは、プログラマーが複雑な接続論理を書いたり保守したりしなくても、サービス、およびサービスによって表されるビジネス機能を接続できるようにする、共通コミュニケーション・インフラストラクチャーを提供します。
- 要求側とサービスの間でのトランスポート・プロトコルの変換。エンタープライズ・サービス・バスは、さまざまな IT 標準を使用するビジネス機能を統合するための、一貫性のある標準ベースの方法を提供します。これによって、通常は通信できないビジネス機能の統合が可能になり、部署内のアプリケーションを接続したり、異なる企業内のアプリケーションをサービス対話に参加させることができるようになります。
- 要求側とサービスの間でのメッセージ・フォーマットの変換。エンタープライズ・サービス・バスは、ビジネス機能間に異なる形式の情報があっても、その交換を可能にします。つまりバスは、あるビジネス機能に配信される情報を、そのアプリケーションで要求される形式で配信されるようにします。
- 本質的に異なるソースからのビジネス・イベントの処理。エンタープライズ・サービス・バスは、サービス要求を処理するためのメッセージ交換に加えて、イベント・ベースの対話をサポートします。

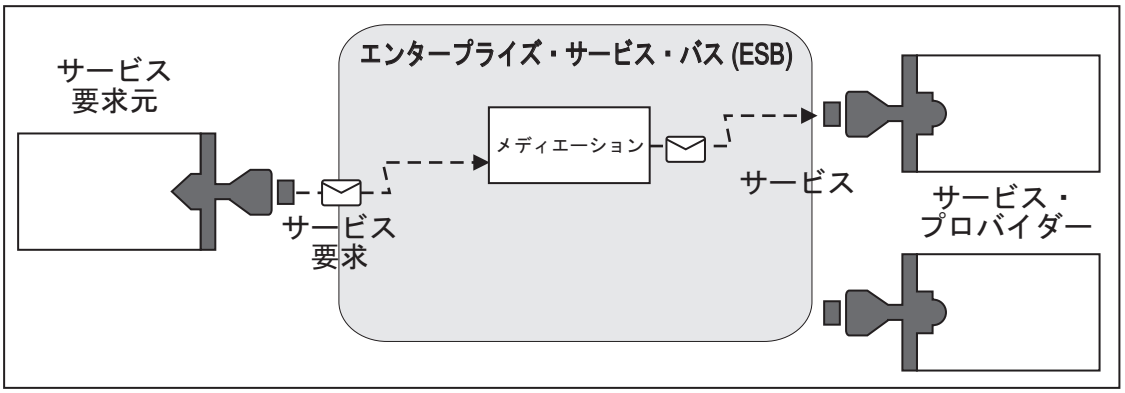


図 17. エンタープライズ・サービス・バス： エンタープライズ・サービス・バスは、サービスの要求側またはプロバイダーであるアプリケーション間で、メッセージをルーティングしています。このバスはトランスポート・プロトコルを変換し、要求側とプロバイダーの間でメッセージ・フォーマットを変換しています。この図では、それぞれのアプリケーションは、(コネクターの異なる形状で表されている) 異なるプロトコルを使用し、異なるメッセージ・フォーマットを使用します。

エンタープライズ・サービス・バスを使用することによって、コンピューター・システムではなく中核業務に集中することができます。必要な場合には、(例えば、ビジネス要件の変更に対応したり、追加のサービス能力を追加したり、新しい機能を追加したりするために) サービスを変更または追加することができます。バスを再構成することによって、バスを使用している既存のサービスやアプリケーションにほとんどあるいはまったく影響を与えることなく、必要な変更を行うことができます。

エンタープライズ・サービス・バスのメッセージング・インフラストラクチャー

WebSphere Process Server には、エンタープライズ・サービス・バス機能があります。WebSphere Process Server は、サービス指向、メッセージ指向、およびイベン

ト・ドリブンの各テクノロジーの統合をサポートすることにより、統合化されたエンタープライズ・サービス・バスにおいて標準ベースのメッセージング・インフラストラクチャーを実現します。

エンタープライズ・アプリケーションで使用可能なエンタープライズ・サービス機能では、サービスの対話を円滑にするために、トランスポート層だけでなく、メディエーション・サポートも提供されます。エンタープライズ・サービス・バスは、オープン・スタンダードおよびサービス指向アーキテクチャー (SOA) を中心に構築されます。これは堅固な Java EE インフラストラクチャーおよび IBM WebSphere Application Server によって提供される関連プラットフォーム・サービスに基づいています。

WebSphere Process Server には、IBM WebSphere Enterprise Service Bus で利用可能なテクノロジーと同じテクノロジーが採用されています。この機能は、WebSphere Process Server の基盤機能であるため、この機能を利用するために WebSphere Enterprise Service Bus 用の追加ライセンスを取得する必要はありません。

ただし、WebSphere Enterprise Service Bus の追加スタンドアロン・ライセンスを企業にデプロイすることで、WebSphere Process Server によるプロセス統合ソリューションの接続性の範囲を拡大することができます。例えば、WebSphere Enterprise Service Bus を SAP アプリケーションのさらに近くにインストールして、IBM WebSphere Adapter for SAP をホストしたり、SAP メッセージを変換してからネットワーク間でその情報を、WebSphere Process Server によって連携可能になったビジネス・プロセスに送信したりすることができます。

WebSphere Enterprise Service Bus を企業にデプロイすることで、フェデレーテッド ESB の一部として別途インストールした WebSphere Process Server またはその他の統合ソリューションの能力を活用して、プロセス統合ソリューションの接続性の範囲を拡大することができます。例えば、WebSphere Enterprise Service Bus を SAP アプリケーションのさらに近くにインストールして、IBM WebSphere Adapter for SAP をホストしたり、SAP メッセージを変換してからネットワーク間でその情報を、WebSphere Process Server によって連携可能になったビジネス・プロセスに送信したりすることができます。

メッセージング宛先ホストまたはキュー宛先ホスト

メッセージング宛先ホストまたはキュー宛先ホストにより、サーバー内にメッセージング機能が提供されます。サーバーをメッセージング・ターゲットとして構成すると、サーバーはメッセージング宛先ホストになります。

メッセージング・エンジンはサーバーの内部で稼働します。メッセージング・エンジンにより、メッセージング機能のほか、アプリケーションがバスに接続するための接続ポイントも提供されます。Service Component Architecture (SCA) の非同期通信、JMS のインポートおよびエクスポート、および非同期の内部処理は、メッセージング・エンジンでメッセージ・キューを使用します。

アプリケーション・モジュールがデプロイされると、デプロイメント環境によりバスを介してメッセージ送信元がメッセージ宛先に関連付けられます。メッセージ送信元とメッセージ宛先を認識すれば、どのようなタイプのデプロイメント環境が必要かを判断するために役立ちます。

アプリケーションは、データベースまたはスキーマ内のテーブルの集合であるデータ・ストア、またはファイル・ストア内に永続データを格納できます。メッセージング・エンジンは JDBC データ・ソースのインスタンスを使用して、そのデータベースと対話します。

管理コンソールの「**サーバー**」を使用してデプロイメント環境を定義するときにメッセージング宛先ホストを構成するか、ソフトウェアのインストール時にサーバーを宛先ホストとして指定します。

データ・ストア:

すべてのメッセージング・エンジンでは、データ・ストアを使用できます。データ・ストアとは、データベースまたはスキーマ内のテーブル一式であり、永続データが格納されます。

データ・ストア内のすべてのテーブルは、同じデータベース・スキーマ内に保持されます。各データ・ストアは、別個のデータベース内に作成することができます。また、同一のデータベース内で複数のデータ・ストアを作成し、それぞれのデータ・ストアに異なるスキーマを使用することもできます。

メッセージング・エンジンは、JDBC データ・ソースのインスタンスを使用して、そのメッセージング・エンジン用のデータ・ストアを含むデータベースと対話します。

データ・ソース

データ・ソースは、アプリケーションとリレーショナル・データベースの間のリンクを提供します。

アプリケーションは、データ・ソースを使用して、リレーショナル・データベースへの接続を取得します。データ・ソースは、他の種類のエンタープライズ情報システム (EIS) に接続するための Java EE コネクター・アーキテクチャー (JCA) 接続ファクトリーに類似しています。

データ・ソースは、JDBC プロバイダーに関連付けられています。JDBC プロバイダーは、特定のタイプのデータベースに対する JDBC 接続に必要なドライバー実装クラスを提供します。アプリケーション・コンポーネントは、データ・ソースを直接操作して、データベースに対する接続インスタンスを取得します。各データ・ソースに対応する接続プールによって、接続を管理できます。

さまざまな設定で複数のデータ・ソースを作成し、それらを同じ JDBC プロバイダーに関連付けることも可能です。例えば、同じデータベース・アプリケーション内でさまざまなデータベースにアクセスするために複数のデータ・ソースを使用する、といった状況が考えられます。WebSphere Process Server では、Sun Microsystems によって定義されている以下のいずれかまたは両方のデータ・ソース・インターフェースを実装するための JDBC プロバイダーが必要です。アプリケーションは、これらのインターフェースを使用して、1 フェーズまたは 2 フェーズのトランザクション・プロトコルで稼働できるようになります。

注: Business Process Choreographer データ・ソースは、Business Process Choreographer 構成ツールを使用して作成されます。Business Process Choreographer の構成を参照してください。

- ConnectionPoolDataSource - 2 フェーズ・コミット・トランザクション以外のローカル・トランザクションとグローバル・トランザクションにアプリケーションが参加することを可能にするデータ・ソース。接続プールのデータ・ソースがグローバル・トランザクションに関わる場合は、トランザクション・マネージャーがトランザクションのリカバリーを実施することはありません。複数のリソース・マネージャーが関わっている場合は、バックアップ・リカバリー・プロセスをアプリケーション側で用意する必要があります。
- XADataSource - 1 フェーズまたは 2 フェーズのトランザクション環境にアプリケーションが参加することを可能にするデータ・ソース。このデータ・ソースがグローバル・トランザクションに関わる場合は、WebSphere Application Server のトランザクション・マネージャーがトランザクションのリカバリーを実施します。

以下の表に、標準的なスタンドアロン環境のセットアップと標準的なデプロイメント環境のセットアップを示します。

表 2. 標準的なスタンドアロン環境のセットアップ

データ・ソース	コンポーネント	有効範囲	JNDI 名
WBI データ・ソース	CommonDB	ノード	jdbc/WPSDB
SCA アプリケーション・バス ME データ・ソース	SCA ME	サーバー	jdbc/com.ibm.ws.sib/nlNode01.server1-SCA.APPLICATION.localhostNode01Cell.Bus
Business Process Choreographer データ・ソース	BPC	サーバー	jdbc/BPEDB
Business Process Choreographer ME データ・ソース	BPC ME	サーバー	jdbc/com.ibm.ws.sib/nlNode01.server1-BPC.localhostNode01Cell.Bus
event	CEI	サーバー	jdbc/cei
CEI ME データ・ソース	CEI ME	サーバー	jdbc/com.ibm.ws.sib/nlNode01.server1-CEI.*cellName*.BUS

表 3. 標準的なデプロイメント環境のセットアップ

データ・ソース	コンポーネント	有効範囲	JNDI 名
WBI データ・ソース	CommonDB	セル	jdbc/WPSDB
SCA アプリケーション・バス ME データ・ソース	SCA ME	クラスター	jdbc/com.ibm.ws.sib/clusterone-SCA.APPLICATION.enduranceTestCell01.Bus
Business Process Choreographer データ・ソース	BPC	クラスター	jdbc/BPEDB
Business Process Choreographer ME データ・ソース	BPC ME	クラスター	jdbc/com.ibm.ws.sib/clusterone-BPC.enduranceTestCell01.Bus

表 3. 標準的なデプロイメント環境のセットアップ (続き)

データ・ソース	コンポーネント	有効範囲	JNDI 名
event	CEI	クラスター	jdbc/cei
CEI ME データ・ソース	CEI ME	クラスター	jdbc/com.ibm.ws.sib/clusterone-CEI.*cellName*.BUS

データ・ソースについて詳しくは、『データ・ソース』を参照してください。WebSphere Application Server インフォメーション・センターの

JDBC プロバイダー:

JDBC プロバイダーを使用することにより、アプリケーションがリレーショナル・データベースと対話できます。

アプリケーションは、JDBC プロバイダーを使用してリレーショナル・データベースと対話します。JDBC プロバイダーには、特定のデータベースのタイプにアクセスするための特定の JDBC ドライバー実装クラスが用意されています。そのデータベースへの接続プールを作成するには、データ・ソースを JDBC プロバイダーに関連付けます。JDBC プロバイダーとデータ・ソース・オブジェクトを組み合わせて実行できる機能は、非リレーショナル・データベースに接続するための Java EE コネクター・アーキテクチャー (JCA) 接続ファクトリーの機能と同等になります。

標準的なスタンドアロン環境のセットアップと標準的なデプロイメント環境のセットアップの両方の例については、前のトピックを参照してください。

JDBC プロバイダーについて詳しくは、WebSphere Application Server インフォメーション・センターの『JDBC プロバイダー』を参照してください。

WebSphere Process Server 用のサービス統合バス

サービス統合バスとは、同期および非同期メッセージングによってサービス統合をサポートする、管理された通信メカニズムです。バスは、バス・リソースを管理する相互接続メッセージング・エンジンで構成されます。サービス統合バスは、WebSphere Process Server の基盤となる WebSphere Application Server テクノロジーの 1 つです。

一部のバスは、システム、デプロイする Service Component Architecture (SCA) アプリケーション、およびその他のコンポーネントで使用するために自動的に作成されます。また、サービス統合論理または他のアプリケーションをサポートするためにバスを作成することもできます。例えば、WebSphere Process Server 内でサービス要求元およびサービス・プロバイダーとして機能するアプリケーションをサポートしたり、WebSphere MQ にリンクしたりするためにバスを作成します。

バス宛先は、アプリケーションがプロデューサー、コンシューマー、またはその両方として接続できる論理アドレスです。キュー宛先は、point-to-point メッセージングに使用するバス宛先です。

各バスは 1 つ以上のバス・メンバーを持ち、各バス・メンバーはサーバーまたはクラスターのいずれかです。

バス・トポロジー とは、アプリケーション・サーバー、メッセージング・エンジン、および WebSphere MQ のキュー・マネージャーの物理的な配置とそれらの間のバス接続およびリンクのパターン (エンタープライズ・サービス・バス を構成する) を意味します。

いくつかのサービス統合バスは、WebSphere Process Server をサポートするために自動的に作成されます。デプロイメント環境を作成するか、または SCA アプリケーションをサポートするためにサーバーまたはクラスターを構成する際に、最大 4 つのバスが作成されます。これらのバスには、それぞれ構成する必要がある 3 つの認証別名があります。

SCA システム・バス:

SCA システム・バス とは、Service Component Architecture (SCA) モジュールのキュー宛先をホストするために使用するサービス統合バスのことです。メディエーション・モジュールをサポートする SCA ランタイムは、システム・バス上のキュー宛先をインフラストラクチャーとして使用して、コンポーネントとモジュール間の非同期対話をサポートします。

システム・バスは、デプロイメント環境を作成する際、または SCA アプリケーションをサポートするためにサーバーまたはクラスターを構成する際に自動的に作成されます。システム・バスは、リソース (キュー宛先など) をメディエーション・モジュールおよび対話エンドポイントに対して構成するスコープを提供します。バスにより、エンドポイント間のメッセージ・ルーティングが使用可能になります。優先順位および信頼性を含む、バスのサービス品質を指定することができます。

バス名は SCA.SYSTEM.busID.Bus です。このバスの保護のために使用される認証別名は、SCA_Auth_Alias です。

SCA アプリケーション・バス:

アプリケーション・バス宛先では、WebSphere Business Integration Adapters と他の System Component Architecture コンポーネントとの間の非同期通信がサポートされています。

アプリケーション・バスは、デプロイメント環境を作成する際、または SCA アプリケーションをサポートするためにサーバーまたはクラスターを構成する際に自動的に作成されます。アプリケーション・バスは、サービス統合論理または他のアプリケーションをサポートするためにユーザーが作成できるサービス統合バスに類似しています。

バス名は SCA.APPLICATION.busID.Bus です。このバスの保護のために使用される認証別名は、SCA_Auth_Alias です。

Common Event Infrastructure バス:

Common Event Infrastructure バスは、構成済みの Common Event Infrastructure サーバーに Common Base Event を非同期に伝送するために使用されます。

バス名は CommonEventInfrastructure_Bus です。このバスの保護のために使用される認証別名は、CommonEventInfrastructureJMSAuthAlias です。

Business Process Choreographer バス:

Business Process Choreographer のバス名と認証を内部メッセージの送信に使用します。

Business Process Choreographer バスは、メッセージの内部的な送信、および Business Flow Manager の Java Messaging Service (JMS) API に使用されます。

バス名は BPC.cellName.Bus です。認証別名は BPC_Auth_Alias です。

サービス・アプリケーションおよびサービス・モジュール

サービス・モジュールは、実行時にサービスを提供する Service Component Architecture (SCA) モジュールです。サービス・モジュールを WebSphere Process Server にデプロイするときには、Enterprise ARchive (EAR) ファイルとしてパッケージされた関連サービス・アプリケーションをビルドします。

サービス・モジュールはデプロイメントの基本単位であり、関連したサービス・アプリケーションが使用するコンポーネント、ライブラリー、およびステージング・モジュールを含めることができます。サービス・モジュールにはエクスポート (オプションでインポートも) が格納され、モジュールとサービス・リクエスター/サービス・プロバイダー間の関係が定義されます。WebSphere Process Server は、ビジネス・サービスのモジュールおよびメディエーション・モジュールをサポートします。モジュールとメディエーション・モジュールは、いずれも SCA モジュールの一種です。メディエーション・モジュールは、サービス起動をターゲットが理解する形式に変換し、要求をターゲットに渡して結果をオリジネーターに戻すことによって、アプリケーション間の通信を可能にします。ビジネス・サービス用のモジュールは、ビジネス・プロセスのロジックを実装します。ただし、メディエーション・モジュール内にパッケージ可能なものと同じメディエーション・ロジックをモジュールに格納することもできます。

サービス・アプリケーションのデプロイ

サービス・アプリケーションを含む EAR ファイルをデプロイするプロセスは、任意の EAR ファイルをデプロイするプロセスと同じです。メディエーション・パラメーターの値はデプロイメント時に変更できます。SCA モジュール が格納された EAR ファイルをデプロイすると、サービス・アプリケーションとそれに関連するモジュールの詳細を表示することができます。サービス・モジュールがエクスポートを介してサービス・リクエスターにどのように接続されているか、およびインポートを介してサービス・プロバイダーにどのように接続されているかを表示することができます。

SCA モジュールの詳細の表示

表示可能なサービス・モジュールの詳細は、 SCA モジュールによって異なります。サービス・モジュールの詳細には、以下の属性を定義することができます。

- SCA モジュール名
- SCA モジュールの説明
- 関連したアプリケーション名
- SCA モジュール バージョン情報 (モジュールがバージョン管理されている場合)

- SCA モジュール・インポート:
  - インポート・インターフェースは、SCA モジュールがサービスにアクセスする方法を記述した抽象定義です。
  - インポート・バインディングは、SCA モジュールがサービスにアクセスするときの物理メカニズムを指定する具象定義です。例えば、SOAP/HTTP を使用します。
- SCA モジュール・エクスポート:
  - エクスポート・インターフェースは、サービス要求元が SCA モジュールにアクセスする方法を記述した抽象定義です。
  - エクスポート・バインディングは、サービス要求元が SCA モジュールにアクセスする (それによって間接的にサービスにアクセスする) ときの物理メカニズムを指定する具象定義です。
- SCA モジュール・プロパティー

インポートとインポート・バインディング

インポートは、Service Component Architecture (SCA) モジュールとサービス・プロバイダー間の対話を定義します。SCA モジュールは、インポートを使用することにより、コンポーネントがローカルの表記を使用して外部サービス (SCA モジュールの外部にあるサービス) にアクセスすることを許可します。インポート・バインディングでは、外部サービスにアクセスする特定の方法を定義します。

SCA モジュールは、外部サービスにアクセスする必要がない場合は、インポートを持つ必要はありません。メディエーション・モジュールには、通常、意図するターゲットにメッセージまたは要求を渡すために使用するインポートが 1 つ以上あります。

インターフェースとバインディング

SCA モジュール・インポートには少なくとも 1 つのインターフェースが必要で、1 つの SCA モジュール・インポートには単一のバインディングがあります。

- インポート・インターフェースは、Web サービスを記述するための XML 言語である Web サービス記述言語 (WSDL) を使用して、操作設定を定義する抽象定義です。1 つの SCA モジュールは、多くのインポート・インターフェースを持つことができます。
- インポート・バインディングとは、SCA モジュールが外部サービスにアクセスするために使用する物理メカニズムを指定する具象定義です。

サポートされるインポート・バインディング

WebSphere Process Server は、以下のインポート・バインディングをサポートしています。

- SCA バインディングでは、SCA モジュールを他の SCA モジュールに接続します。SCA バインディングはデフォルト・バインディングとも呼ばれます。
- Web サービス・バインディングにより、コンポーネントが Web サービスを起動することが許可されます。サポートされるプロトコルは SOAP1.1/HTTP、SOAP1.2/HTTP および SOAP1.1/JMS です。

Java API for XML Web Services (JAX-WS) をベースとした SOAP1.1/HTTP または SOAP1.2/HTTP バインディングが使用できます。これにより、文書または RPC リテラル・バインディングを用いてサービスとの対話が可能になります。呼び出しのカスタマイズには、JAX-WS ハンドラーを使います。個別に SOAP1.1/HTTP バインディングが提供され、RPC エンコードされたバインディングを使用するサービスを使った対話や、呼び出しのカスタマイズに JAX-RPC ハンドラーを使用する必要のある場合の対話が可能となります。

- HTTP バインディングでは、HTTP プロトコルを使用したアプリケーションへのアクセスが許可されます。
- Enterprise JavaBeans (EJB) インポート・バインディングにより、SCA コンポーネントは、Java EE サーバー上で実行される Java EE ビジネス・ロジックで提供されるサービスを起動することができます。
- エンタープライズ情報システム (EIS) のバインディングは、SCA コンポーネントと外部 EIS 間の接続を提供します。この通信はリソース・アダプターを使って行います。
- Java Message Service (JMS) 1.1 バインディングにより、WebSphere Application Server のデフォルト・メッセージング・プロバイダーとの相互運用が許可されます。JMS は、TCP/IP や HTTP または HTTPS などのさまざまなトランスポート・タイプを活用します。JMS Message クラスおよびその 5 つのサブタイプ (Text、Bytes、Object、Stream、および Map) が自動的にサポートされます。
- 汎用 JMS バインディングにより、JMS Application Server Facility (ASF) を使用して WebSphere Application Server を統合するサード・パーティーの JMS プロバイダーとの相互運用が許可されます。
- WebSphere MQ JMS バインディングにより、WebSphere MQ ベースの JMS プロバイダーとの相互運用が可能になります。JMS Message クラスおよびその 5 つのサブタイプ (Text、Bytes、Object、Stream、および Map) が自動的にサポートされます。WebSphere MQ を JMS プロバイダーとして使用する場合は、WebSphere MQ JMS バインディングを使います。
- WebSphere MQ バインディングにより、WebSphere MQ との相互運用が許可されます。WebSphere MQ バインディングは、WebSphere MQ クライアント接続を介したリモート・キュー・マネージャーとの間でのみ使用できます。ローカル・キュー・マネージャーとの間では使用できません。ネイティブの WebSphere MQ アプリケーションと通信する場合は、WebSphere MQ バインディングを使用します。

サービスの動的呼び出し

サービスは、サポートされるどのインポート・バインディングを使用しても呼び出すことができます。通常の場合、サービスはインポート内で指定されたエンドポイントにあります。このエンドポイントは静的エンドポイントと呼ばれます。静的エンドポイントを指定変更することで、異なるサービスを呼び出すことも可能です。静的エンドポイントを動的に指定変更すれば、サポートされるあらゆるインポート・バインディングを使用して、別のエンドポイントでサービスを呼び出すことができます。またサービスの動的起動によって、サポートされるインポート・バインディングが静的エンドポイントを持たない場所でサービスを呼び出すことができます。

動的起動のためのプロトコルと構成を指定するには、関連づけられたバインディングを用いたインポートを使用します。動的起動で使用するインポートを呼び出し側のコンポーネントと接続するか、実行時に動的に選択することができます。

Web サービスと SCA 呼び出しでは、エンドポイント URL から推定したプロトコルと構成を使って、インポートせずに動的起動を行うことも可能です。呼び出しターゲットのタイプはエンドポイント URL から識別します。インポートを使う場合は、URL はインポート・バインディングのプロトコルと互換性がある必要があります。

- SCA URL には、他の SCA モジュールの呼び出しが指示されています。
- デフォルトの HTTP または JMS URL には、Web サービスの呼び出しが指示されています。これらの URL には、その URL が HTTP または JMS バインディングを利用した呼び出しを表していることを示す、追加のバインディング・タイプの値を記述することが可能です。
- Web サービス HTTP URL では、デフォルトで SOAP 1.1 を使用する設定になっていますが、SOAP 1.2 の使用を指定するバインディング・タイプの値を指定することができます。

エクスポートとエクスポート・バインディング

エクスポートは、Service Component Architecture (SCA) モジュールとサービス要求元間の対話を定義します。SCA モジュールはエクスポートを使用して、他のモジュールにサービスを提供します。エクスポート・バインディングは、SCA モジュールがサービス要求元によってアクセスされる際の特定の方法を定義します。

インターフェースとバインディング

SCA モジュールのエクスポートには、少なくとも 1 つのインターフェースが必要です。

- エクスポート・インターフェースは、Web サービスを記述するための XML 言語である Web サービス記述言語 (WSDL) を使用して、操作設定を定義する抽象定義です。1 つの SCA モジュールは、多くのエクスポート・インターフェースを持つことができます。
- エクスポート・バインディングとは、サービス要求元がサービスにアクセスするために使用する物理メカニズムを指定する具象定義です。通常、SCA モジュールのエクスポートには、1 つのバインディングが指定されています。バインディングが指定されていないエクスポートは、SCA バインディングを持つエクスポートとしてランタイムによって解釈されます。

サポートされるエクスポート・バインディング

WebSphere Process Server は、以下のエクスポート・バインディングをサポートしています。

- SCA バインディングでは、SCA モジュールを他の SCA モジュールに接続します。SCA バインディングはデフォルト・バインディングとも呼ばれます。
- Web サービス・バインディングにより、エクスポートを Web サービスとして起動することが許可されます。サポートされるプロトコルは SOAP1.1/HTTP、SOAP1.2/HTTP および SOAP1.1/JMS です。

Java API for XML Web Services (JAX-WS) をベースとした SOAP1.1/HTTP または SOAP1.2/HTTP バインディングが使用できます。これにより、文書または RPC リテラル・バインディングを用いてサービスとの対話が可能になります。呼び出しのカスタマイズには、JAX-WS ハンドラーを使います。個別に SOAP1.1/HTTP バインディングが提供され、RPC エンコードされたバインディングを使用するサービスを使った対話や、呼び出しのカスタマイズに JAX-RPC ハンドラーを使用する必要のある場合の対話が可能となります。

- HTTP バインディングにより、HTTP プロトコルを使用してエクスポートへのアクセスが許可されます。
- Enterprise JavaBeans (EJB) のエクスポート・バインディングを使うと、他の方法では利用できない場合でも Java EE ビジネス・ロジックが SCA コンポーネントを呼び出せるように、SCA コンポーネントを EJB として公開できるようになります。
- エンタープライズ情報システム (EIS) のバインディングは、SCA コンポーネントと外部 EIS 間の接続を提供します。この通信はリソース・アダプターを使って行います。
- Java Message Service (JMS) 1.1 バインディングにより、WebSphere Application Server のデフォルト・メッセージング・プロバイダーとの相互運用が許可されます。JMS は、TCP/IP や HTTP または HTTPS などのさまざまなトランスポート・タイプを活用します。JMS Message クラスおよびその 5 つのサブタイプ (Text、 Bytes、Object、Stream、および Map) が自動的にサポートされます。
- 汎用 JMS バインディングにより、JMS Application Server Facility (ASF) を使用して WebSphere Application Server を統合するサード・パーティーの JMS プロバイダーとの相互運用が許可されます。
- WebSphere MQ JMS バインディングにより、WebSphere MQ ベースの JMS プロバイダーとの相互運用が可能になります。JMS Message クラスおよびその 5 つのサブタイプ (Text、Bytes、Object、Stream、および Map) が自動的にサポートされます。WebSphere MQ を JMS プロバイダーとして使用する場合は、WebSphere MQ JMS バインディングを使います。
- WebSphere MQ バインディングにより、WebSphere MQ との相互運用が許可されます。リモート (つまりクライアント) 接続を使って、リモート・マシン上の MQ キュー・マネージャーに接続します。ローカル (つまりバインディング) 接続とは、WebSphere MQ への直接接続のことです。これは同じマシン上の MQ キュー・マネージャーへの接続にのみ使用できます。WebSphere MQ では、両方のタイプの接続が許可されますが、MQ バインディングでは、「リモート」(つまり「クライアント」) 接続のみがサポートされます。

メディエーション・モジュール

メディエーション・モジュールは、サービス要求のフォーマット、内容、またはターゲットを変更できる Service Component Architecture (SCA) モジュールです。

メディエーション・モジュールは、サービス要求元とサービス・プロバイダー間で伝送中のメッセージを操作します。メッセージを異なるサービス・プロバイダーにルーティングでき、またメッセージの内容または形式を修正することもできます。メディエーション・モジュールは、ユーザーの要件に合わせて調整したメッセージ・ロギングやエラー処理などの機能を提供できます。

メディエーション・モジュールの一定の側面を、モジュールを再デプロイすることなく、WebSphere Process Server 管理コンソールから変更できます。

メディエーション・モジュールのコンポーネント

メディエーション・モジュールには、以下のアイテムが含まれています。

- インポート。SCA モジュールとサービス・プロバイダー間の対話を定義します。これにより、SCA モジュールは、外部サービスをローカル・サービスのように呼び出すことができます。WebSphere Process Server を使用してメディエーション・モジュール・インポートを表示し、バインディングを変更できます。
- エクスポート。SCA モジュールとサービス・リクエスター間の対話を定義します。これにより、SCA モジュールでサービスを提供し、SCA モジュールの外部インターフェース (アクセス・ポイント) を定義することが可能になります。メディエーション・モジュール・エクスポートは、WebSphere Process Server から表示できます。
- SCA コンポーネント。メディエーション・モジュールなどの SCA モジュール用のビルディング・ブロックです。 SCA モジュールとコンポーネントは、WebSphere Integration Developer を使用してグラフィカルに作成およびカスタマイズできます。メディエーション・モジュールのデプロイ後に、WebSphere Process Server の管理コンソールから、モジュールを再デプロイせずにその属性をカスタマイズできます。

  一般に、メディエーション・モジュールは、*メディエーション・フロー・コンポーネント* と呼ばれる特定のタイプの SCA コンポーネントを含みます。メディエーション・フロー・コンポーネントは、メディエーション・フローを定義します。

  メディエーション・フロー・コンポーネントには、メディエーション・プリミティブを 1 つまたは複数含めることができますが、まったく含めないことも可能です。WebSphere Process Server では、メッセージ・ルーティングおよび変換のための機能を提供する、あらかじめ用意された 1 組のメディエーション・プリミティブがサポートされます。メディエーション・プリミティブをより柔軟に使用する必要がある場合は、カスタム・メディエーション・プリミティブを使用して、カスタム・ロジックを呼び出すことができます。

  メディエーション・フロー・コンポーネントが含まれていないメディエーション・モジュールには、サービス要求をあるプロトコルから別のプロトコルに変換するという目的があります。例えば、SOAP/JMS を使用して作成したサービス要求を、送信前に SOAP/HTTP に変換しなければならない場合があります。

  注: WebSphere Process Server から、メディエーション・モジュールを表示して、一定の変更を加えることができます。ただし、WebSphere Process Server モジュールの内部から SCA コンポーネントを表示あるいは変更することはできません。SCA コンポーネントをカスタマイズするには、WebSphere Integration Developer を使用してください。

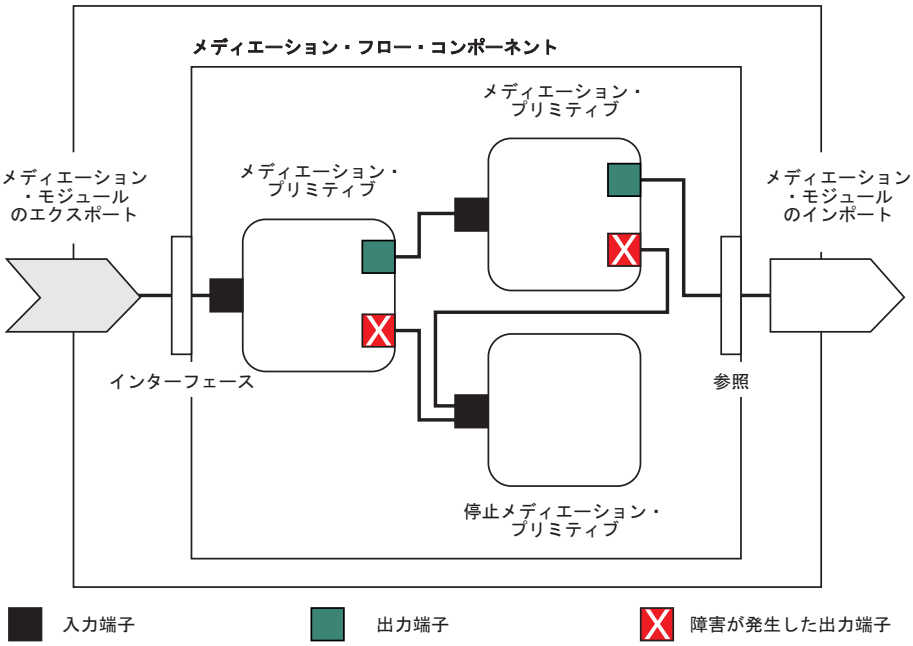


図 18. メディエーション・モジュールの簡単な例： メディエーション・モジュールには、メディエーション・プリミティブが含まれた 1 つのメディエーション・フロー・コンポーネントが含まれています。

- プロパティー

  メディエーション・プリミティブはプロパティーを持ちます。一部のプロパティーは、SCA モジュールの追加のプロパティーとして管理コンソールに表示させることができます。

  メディエーション・プリミティブのプロパティーを WebSphere Process Server 管理コンソールで表示するには、統合開発者はこれらのプロパティーをプロモートする必要があります。一部のプロパティーは管理上の構成に適しています。これらのプロパティーは統合サイクルから管理サイクルへプロモート可能であるため、WebSphere Integration Developer はこれらのプロパティーをプロモート可能なプロパティーとして記述します。他のプロパティーは、変更するとメディエーション・フローに影響が生じ、メディエーション・モジュールの再デプロイが必要になるおそれがあるため、管理構成には適しません。WebSphere Integration Developer は、メディエーション・プリミティブのプロモートされるプロパティーの下に、プロモート対象として選択できるプロパティーのリストを示します。

  WebSphere Process Server 管理コンソールを使用すると、メディエーション・モジュールの再デプロイや、サーバーまたはモジュールの再始動を行うことなく、プロモートされるプロパティーの値を変更できます。

  一般的に、メディエーション・フローはプロパティーの変更を即時に使用します。ただし、デプロイメント・マネージャー・セル内でプロパティーの変更が生じた場合、各ノードで同期化が行われるとそのノード上で変更が有効になります。また、処理中のメディエーション・フローでは、前の値が引き続き使用されます。

注: 管理コンソールではプロパティー値のみ変更でき、プロパティーのグループ、名前、またはタイプは変更できません。プロパティーのグループ、名前、またはタイプを変更する必要がある場合、WebSphere Integration Developer を使用する必要があります。

- また、メディエーション・モジュールや従属ライブラリーにより、サブフローを定義することもできます。サブフローにより、統合論理の再使用可能部分として共にワイヤリングされたメディエーション・プリミティブのセットがカプセル化されます。メディエーション・フローにプリミティブを追加して、サブフローを呼び出すことができます。

メディエーション・モジュールのデプロイ

メディエーション・モジュールは、WebSphere Integration Developer を使用して作成され、一般に WebSphere Process Server のエンタープライズ・アーカイブ (EAR) ファイル内にデプロイされます。

プロモートされるプロパティーの値は、デプロイメント時に変更できます。

WebSphere Integration Developer からメディエーション・モジュールをエクスポートすると、WebSphere Integration Developer にメディエーション・モジュールを Java アーカイブ (JAR) ファイル内にパッケージ化させ、その JAR ファイルを EAR ファイル内にパッケージ化させることができます。この後、管理コンソールから新規アプリケーションをインストールすることにより、EAR ファイルをデプロイできます。

メディエーション・モジュールは 1 つのエンティティーと考えられます。ただし、SCA モジュールは、1 つの JAR ファイルに保管されたいくつかの XML ファイルによって定義されます。

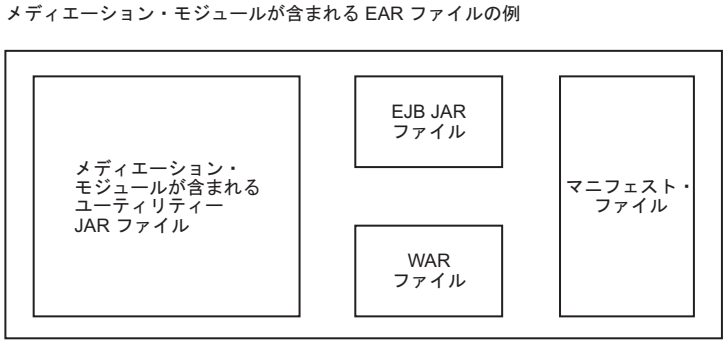


図 19. メディエーション・モジュールが含まれる EAR ファイルの簡単な例: EAR ファイルには JAR が含まれます。ユーティリティー JAR ファイルにはメディエーション・モジュールが含まれます。

メディエーション・プリミティブ

メディエーション・フロー・コンポーネントは、サービス・コンポーネント間のメッセージ・フローを操作します。メディエーション・コンポーネントの機能は、標準サービス実装タイプをインプリメントするメディエーション・プリミティブ によってインプリメントされます。

メディエーション・フロー・コンポーネントには、1 つ以上のフローがあります。例えば、要求用のフロー、応答用のフローなどです。

WebSphere Process Server は、提供されたメディエーション・プリミティブのセットをサポートしており、このメディエーション・プリミティブ・セットは、WebSphere Process Server にデプロイされたメディエーション・モジュールまたはモジュールの標準メディエーション機能を実装します。特殊なメディエーション機能が必要な場合は、ユーザー独自のカスタム・メディエーション・プリミティブを開発できます。

メディエーション・プリミティブは、サービス・メッセージ・オブジェクト (SMO) で表されるメッセージを処理するかまたは取り扱う 1 つの「入力」操作を定義します。メディエーション・プリミティブは、メッセージを別のコンポーネントまたはモジュールに送信する「出力」操作も定義します。

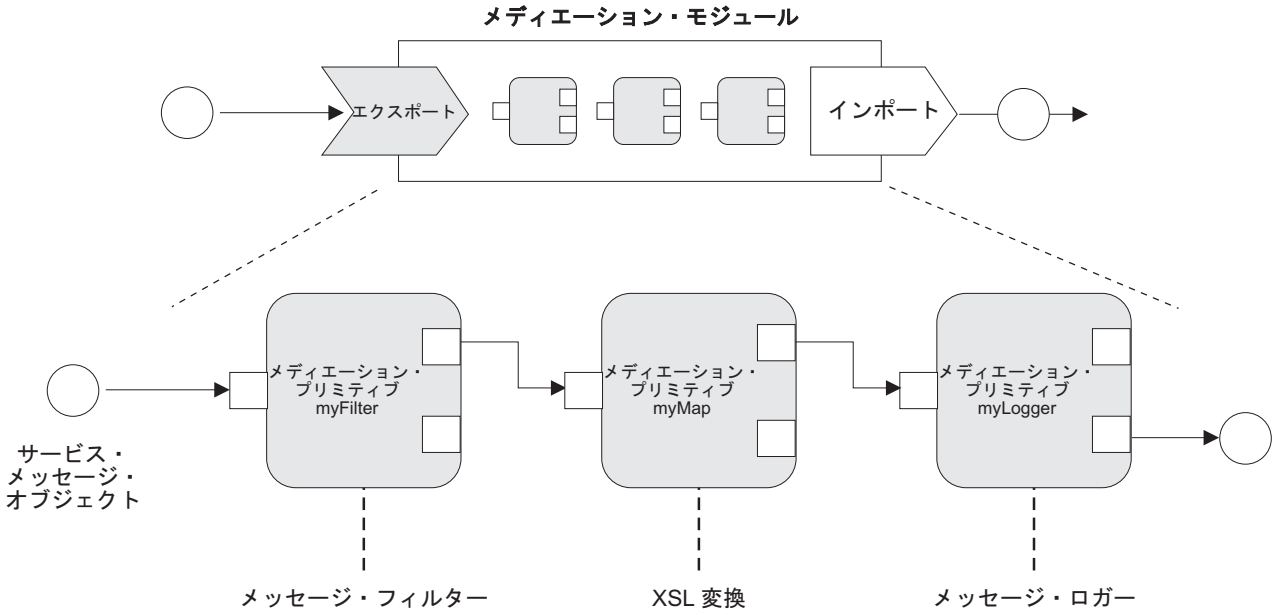


図 20. 3 つのメディエーション・プリミティブから成るメディエーション・モジュール

WebSphere Integration Developer を使用すると、メディエーション・プリミティブを構成して、そのプロパティーを設定できます。これらのプロパティーの一部は、プロモートを行うことにより、ランタイム管理者に対して表示することができます。プロモート可能なメディエーション・プリミティブ・プロパティーは、すべて動的プロパティーにすることもできます。動的プロパティーは、ポリシー・ファイルを使用して実行時にオーバーライドすることができます。

WebSphere Integration Developer を使用すると、メディエーション・プリミティブを基にしてメディエーション・フロー・コンポーネントをグラフィカルにモデル化して組み立てることと、メディエーション・フロー・コンポーネントを基にメディエーション・モジュールまたはモジュールを組み立てることができます。管理コンソールではメディエーション・モジュールおよびモジュールのことを SCA モジュールと呼びます。

WebSphere Integration Developer により、モジュールやモジュールの従属ライブラリー内にサブフローを定義することもできます。サブフローには、ポリシー解決メディエーション・プリミティブを除くすべてのメディエーション・プリミティブを定義することができます。サブフローは、要求フローや応答フロー、またはサブフロー・メディエーション・プリミティブを使用する別のサブフローから呼び出されます。サブフロー内のメディエーション・プリミティブからプロモートされたプロパティーは、サブフロー・メディエーション・プリミティブ上のプロパティーとして公開されます。このプロパティーは、ランタイム管理者による変更が可能なモジュール・レベルに到達するまで、再度プロモートすることができます。

サポートされるメディエーション・プリミティブ

以下のメディエーション・プリミティブ・セットが WebSphere Process Server でサポートされています。

ビジネス・オブジェクト・マップ
メッセージを変換します。

- 再利用可能なビジネス・オブジェクト・マップを使用して、メッセージ変換を定義します。
- ビジネス・オブジェクト・マップ・エディターを使用して、メッセージ変換をグラフィカルに定義できます。
- メッセージの内容を変更できます。
- 入力メッセージ・タイプを別の出力メッセージ・タイプに変換できます。

カスタム・メディエーション
独自のメディエーション・ロジックを Java コードに実装できます。カスタム・メディエーション・プリミティブは、ユーザー定義メディエーション・プリミティブの柔軟性と、事前定義メディエーション・プリミティブの簡素性を兼ね備えています。以下の方法により、複雑な変換およびルーティング・パターンを作成できます。

- Java コードの作成。
- 独自プロパティーの作成。
- 新規端末の追加。

カスタム・メディエーション・プリミティブからサービスを呼び出すことは可能ですが、サービス起動メディエーション・プリミティブはサービスを呼び出す目的で設計されており、再試行などの追加機能を提供します。

データ・ハンドラー
メッセージの一部を変換できます。メッセージのエレメントを物理形式から論理構造に変換したり、論理構造から物理形式に変換したりするために使用します。プリミティブの主な使用目的は、JMS テキスト・メッセージ・オブジェクト内のテキスト・ストリングなどの物理形式を論理ビジネス・オブ

ジェクト構造に、およびその逆方向に変換することです。このメディエーションは、一般に次の目的で使用されます。

- 入力メッセージのセクションを定義済み構造から別の構造に変換する - 一例として、SMO にコンマ区切りのストリング値が含まれており、この SMO を解析して特定のビジネス・オブジェクトにする場合が挙げられます。
- メッセージ・タイプを変更する - 例えば、JMS エクスポートで JMS 基本タイプト・データ・バインディングを使用するように構成されており、統合開発者がメディエーション・モジュール内でコンテンツを特定の BO 構造まで拡張することを指定する場合などです。

データベース・ルックアップ

ユーザーが提供するデータベースからの情報を使用して、メッセージを変更します。

- データベース・ルックアップ・メディエーション・プリミティブを使用できるように、データベース、データ・ソース、およびサーバー認証の設定をセットアップする必要があります。これを簡単に実行するには、管理コンソールを使用します。
- データベース・ルックアップ・メディエーション・プリミティブは、1 つのテーブルからのみ読み取ることができます。
- 指定されたキー列には、固有な値が含まれていなければなりません。
- 値列のデータは、単純 XML スキーマ・タイプか、単純 XML スキーマ・データを拡張した XML スキーマ・タイプになっている必要があります。

エンドポイント・ルックアップ

リポジトリー内でサービス・エンドポイントを検索することにより、要求の動的ルーティングを可能にします。

- サービス・エンドポイント情報は、WebSphere Service Registry and Repository (WSRR) から取得されます。WSRR レジストリーは、ローカルまたはリモートにすることができます。
- レジストリーの変更を WSRR 管理コンソールから行います。
- WebSphere Process Server は、どのレジストリーを使用するかを認識する必要があるため、WebSphere Process Server 管理コンソールを使用して WSRR アクセス定義を作成する必要があります。

イベント・エミッター

イベントをメディエーション・フロー・コンポーネント内から送信できるようにして、モニター機能を拡張します。

- メディエーション・アクションは、チェック・ボックスを選択解除することによって中断できます。
- この後、WebSphere Process Server 上の Common Base Events (CBE) ブラウザーを使用してイベント・エミッターのイベントを表示することができます。
- パフォーマンス上の理由から、メディエーション・フロー内の重要な点でのみイベントを送信するようにします。
- イベントに記録されるメッセージの一部分を定義できます。

- イベントは Common Base Event の形で、Common Event Infrastructure サーバーに送信されます。
- イベント・エミッター情報を十分活用するには、イベント・コンシューマーが Common Base Event の構造を理解しておく必要があります。Common Base Events には包括的なスキーマがありますが、このスキーマでは拡張データ・エレメントに含まれるアプリケーション固有のデータはモデル化されません。拡張データ・エレメントをモデル化するために、WebSphere Integration Developer のツール群により、構成済みのイベント・エミッター・メディエーション・プリミティブ ごとに Common Event Infrastructure イベント・カタログ定義ファイルが生成されます。イベント・カタログ定義ファイルは、ユーザーのサポート用に提供されるエクスポート成果物ですが、これらは WebSphere Integration Developer や WebSphere Process Server ランタイムでは使用されません。イベント・カタログ定義ファイルは、イベント・エミッターのイベントを利用するアプリケーションを作成するときに参照する必要があります。
- WebSphere Process Server から、その他のモニターを指定することができます。例えば、インポートおよびエクスポートから送信するイベントをモニターすることができます。

障害　フロー内の特定のパスを停止して、例外を生成します。

ファンイン

メッセージを集約 (結合) するのに役立ちます。

- ファンアウト・メディエーション・プリミティブと組み合わせた場合にのみ使用できます。
- ファンアウト・メディエーション・プリミティブとファンイン・メディエーション・プリミティブを組み合わせて使用すると、データを 1 つの出力メッセージに集約できます。
- ファンイン・メディエーション・プリミティブは、決定点に到達するまで複数のメッセージを受信し、1 つのメッセージを出力します。
- 集約データを保持するには、共用コンテキストを使用します。

ファンアウト

メッセージを分割して集約 (結合) するのに役立ちます。

- ファンアウト・メディエーション・プリミティブとファンイン・メディエーション・プリミティブを組み合わせて使用すると、データを 1 つの出力メッセージに集約できます。
- 繰り返しモードでは、ファンアウト・メディエーション・プリミティブにより、繰り返し要素を含む単一の入力メッセージを繰り返すことができます。繰り返し要素が出現するたびに、メッセージが送信されます。
- 集約データを保持するには、共用コンテキストを使用します。

HTTP ヘッダー・セッター

HTTP メッセージ・ヘッダーを管理するためのメカニズムを提供します。

- HTTP メッセージ・ヘッダーを作成、設定、コピー、または削除できます。
- 複数のアクションを設定して複数の HTTP ヘッダーを変更できます。

MQ ヘッダー・セッター

MQ メッセージ・ヘッダーを管理するメカニズムを提供します。

- MQ メッセージ・ヘッダーを作成、設定、コピー、または削除できます。
- 複数のアクションを設定して複数の MQ ヘッダーを変更できます。

SOAP ヘッダー・セッター

SOAP メッセージ・ヘッダーを管理するためのメカニズムを提供します。

- SOAP メッセージ・ヘッダーを作成、設定、コピー、または削除できます。
- 複数のアクションを設定して複数の SOAP ヘッダーを変更できます。

メッセージ・エレメント・セッター

メッセージの内容を設定するための単純な機能を提供します。

- メッセージ・エレメントを変更、追加、または削除することができます。
- メッセージのタイプは変更されません。
- 値列のデータは、単純 XML スキーマ・タイプか、単純 XML スキーマ・データを拡張した XML スキーマ・タイプになっている必要があります。

メッセージ・フィルター

メッセージ内容に基づいて、さまざまなパスにメッセージを送付します。

- メディエーション・アクションは、チェック・ボックスを選択解除することによって中断できます。

メッセージ・ロガー

リレーショナル・データベース内に、またはユーザー独自のカスタム・ロガーによってメッセージをログに記録します。メッセージは XML として保管されるので、データは XML 対応アプリケーションで後処理できます。

- メディエーション・アクションは、チェック・ボックスを選択解除することによって中断できます。
- Rational データベース・スキーマ (テーブル構造) は IBM によって定義されます。
- デフォルトでは、メッセージ・ロガー・メディエーション・プリミティブが共通データベースを使用します。ランタイムは、`jdbc/mediation/messageLog` にあるデータ・ソースを共通データベースにマップします。
- ハンドラー実装クラスを設定して、カスタム・ロガーの動作をカスタマイズできます。オプションとして、フォーマッター実装クラス、フィルター実装クラス、またはその両方を設定して、カスタム・ロガーの動作をカスタマイズすることもできます。

ポリシー解決

リポジトリー内でサービス・エンドポイントおよび関連付けられたポリシー・ファイルを検索することにより、要求の動的構成を可能にします。

- ポリシー・ファイルを使用して、他のメディエーション・プリミティブのプロモートされたプロパティーを動的にオーバーライドできます。
- サービス・エンドポイント情報およびポリシー情報は、WebSphere Service Registry and Repository (WSRR) から取得されます。WSRR レジストリーは、ローカルまたはリモートにすることができます。

- レジストリーの変更を WSRR 管理コンソールから行います。
- WebSphere Process Server は、どのレジストリーを使用するかを認識する必要があるため、WebSphere Process Server 管理コンソールを使用して WSRR アクセス定義を作成する必要があります。

サービス起動

メディエーション・フローが終了するまで待ってからコールアウト・メカニズムを使用するのではなく、メディエーション・フローの内部からサービスを呼び出します。

- サービスから障害が返された場合は、同じサービスを再試行するか、別のサービスを呼び出すことができます。
- サービス起動メディエーション・プリミティブは、単純なサービス呼び出しに対して単独で使用できるだけでなく、複雑なメディエーションの場合は他のメディエーション・プリミティブと組み合わせて使用することもできる強力なメディエーション・プリミティブです。

メッセージ型の設定

統合開発時に、型付けの弱いメッセージ・フィールドを型付けの強いメッセージ・フィールドであるかのように処理できます。フィールドに複数のデータ型を入力できる場合、そのフィールドは型付けの弱いフィールドです。フィールドの型および内部構造が既知の場合、そのフィールドは型付けの強いフィールドです。

- メッセージ型の設定メディエーション・プリミティブを使用すると、予想したデータ型とメッセージの内容が一致していることを実行時に確認できます。

停止　例外を生成せずに、フロー内の特定のパスを停止します。

タイプ・フィルター

タイプに基づいて、フローの別のパスにメッセージを誘導できます。

XSL 変換

メッセージを変換します。

- Extensible Stylesheet Language (XSL) 変換を実行できます。
- XSLT 1.0 変換を使用してメッセージを変換します。この変換は、メッセージの XML 直列化を処理します。

動的ルーティング

統合時に定義されたエンドポイントを使用したり、実行時に動的に判別されたエンドポイントを使用したりして、さまざまな方法でメッセージをルーティングすることができます。

動的ルーティングは、フローは動的だが、考えられるすべてのエンドポイントが Service Component Architecture (SCA) モジュール で事前定義されているメッセージ・ルーティング、およびフローが動的で、エンドポイントの選択も動的であるメッセージ・ルーティングを扱います。後者の場合、サービス・エンドポイントは実行時に外部ソースから選択されます。

動的エンドポイントの選択

ランタイムには、メッセージ・ヘッダー・エレメントによって識別されるエンドポイント・アドレスに要求および応答のメッセージをルーティングする機能があります。このメッセージ・ヘッダー・エレメントは、メディエーション・フロー内の メディエーション・プリミティブによって更新できます。エンドポイント・アドレスは、レジストリーからの情報、データベース、およびメッセージ自体からの情報で更新できます。応答メッセージのルーティングは、応答が Web サービス JAX-WS エクスポートによって送信される場合にのみ適用されます。

要求または応答時に動的ルーティングをランタイムに実装するには、 SCA モジュールで「メッセージ・ヘッダーで設定する場合は動的エンドポイントを使用」プロパティーが設定されている必要があります。統合開発者は、「メッセージ・ヘッダーで設定する場合は動的エンドポイントを使用」プロパティーを設定することができます。または、ランタイム管理者がこのプロパティーを設定できるようにプロモート (実行時に表示) することもできます。「モジュール・プロパティー」ウィンドウでモジュール・プロパティーを表示できます。ウィンドウを表示するには、「**アプリケーション**」 > 「**SCA モジュール**」 > 「**モジュール・プロパティー**」をクリックします。統合開発者はプロモートされたプロパティーに別名を付けます。それらは管理コンソールで表示される名前になります。

レジストリー

IBM WebSphere Service Registry and Repository (WSRR) を使用してサービス・エンドポイント情報を保管し、SCA モジュールを作成して WSRR レジストリーからエンドポイントを取得できます。

SCA モジュールを開発する場合、エンドポイント・ルックアップ・メディエーション・プリミティブを使用して、メディエーション・フローが WSRR レジストリーを照会してサービス・エンドポイントまたはサービス・エンドポイントのセットを調べられるようにできます。SCA モジュールがエンドポイントのセットを取得する場合、別のメディエーション・プリミティブを使用して優先されるものを選択する必要があります。

サービス要求のメディエーション・ポリシー制御

メディエーション・ポリシーを使用して、サービス要求元とサービス・プロバイダー間のメディエーション・フローを制御することができます。

IBM WebSphere Service Registry and Repository (WSRR) に保管されたメディエーション・ポリシーを使用して、メディエーション・フローを制御することができます。WSRR でのサービス・ポリシー管理の実装は、Web サービス・ポリシー・フレームワーク (WS-Policy) に基づいています。

メディエーション・ポリシーを使用してサービス要求を制御するには、適切な Service Component Architecture (SCA) モジュールとメディエーション・ポリシー文書を WSRR レジストリーに備えている必要があります。

メディエーション・ポリシーをサービス要求に付加する方法

メディエーション・ポリシーを使用する必要がある SCA モジュールを開発する場合、ポリシー解決メディエーション・プリミティブをメディエーション・フローに

組み込む必要があります。実行時に、ポリシー解決メディエーション・プリミティブは、メディエーション・ポリシー情報をレジストリーから取得します。したがって、サービス要求のメディエーション・ポリシー制御をサポートするには、SCA モジュールにメディエーション・フロー・コンポーネントが含まれている必要があります。

レジストリー内では、1 つ以上のメディエーション・ポリシーを 1 つの SCA モジュールに付加したり、SCA モジュールで使用されるターゲット・サービスに付加したりすることができます。付加されたメディエーション・ポリシーは、SCA モジュールにより処理されるすべてのサービス・メッセージに使用できます (その有効範囲内にあります)。メディエーション・ポリシーは、条件を定義するポリシー・アタッチメントを持つことができます。メディエーション・ポリシー条件により、さまざまなメディエーション・ポリシーをさまざまなコンテキストで適用できます。さらに、メディエーション・ポリシーには、ガバナンス状態を指定するために使用できる種別を設けることができます。

WebSphere Service Registry and Repository

WebSphere Service Registry and Repository (WSRR) 製品を使用すると、サービス・エンドポイントとメディエーション・ポリシーに関する情報の保管、アクセス、管理を行うことができます。 WSRR によってより動的なサービス・アプリケーションを実現し、絶えず変化するビジネス条件に適応することができます。

概要

メディエーション・フローによって WSRR を動的な検索メカニズムとして使用し、サービス・エンドポイントやメディエーション・ポリシーに関する情報を取得することができます。

WSRR へのアクセスを構成するには、管理コンソールを使用して WSRR 定義文書を作成します。あるいは、wsadmin スクリプト・クライアントから WSRR 管理コマンドを使用できます。WSRR 定義とその接続プロパティーは、レジストリー・インスタンスへの接続と、サービス・エンドポイントやメディエーション・ポリシーの取得に使用されるメカニズムです。

サービス・エンドポイント

WSRR を使用して、既に使用したサービス、これから使用するサービス、知っておくべきサービスに関する情報を保管することができます。これらのサービスの場所は、現在のシステム上でも他のシステム上でもかまいません。例えばアプリケーションは、WSRR を使用して、機能およびパフォーマンスの必要性を満たすもっとも適切なサービスを見つけることができます。

WSRR からサービス・エンドポイントにアクセスする必要がある SCA モジュールを開発する場合、エンドポイント・ルックアップ・メディエーション・プリミティブをメディエーション・フローに組み込む必要があります。エンドポイント・ルックアップ・メディエーション・プリミティブは、実行時にサービス・エンドポイントをレジストリーから取得します。

メディエーション・ポリシー

WSRR を使用して、メディエーション・ポリシー情報を保管することもできます。メディエーション・ポリシーを使用して動的にモジュール・プロパティーをオーバーライドすることにより、サービス要求を制御することができます。SCA モジュールまたはターゲット・サービスを表すオブジェクトに付加されているメディエーション・ポリシーが WSRR に含まれている場合は、そのメディエーション・ポリシーによってモジュールのプロパティーがオーバーライドされる場合があります。異なるコンテキストで別のメディエーション・ポリシーを適用したい場合は、メディエーション・ポリシー条件を作成します。

注: メディエーション・ポリシーはメディエーション・フローの制御に関わるもので、セキュリティーとは関係ありません。

メディエーション・ポリシーを使用する必要がある SCA モジュールを開発する場合、ポリシー解決メディエーション・プリミティブをメディエーション・フローに組み込む必要があります。実行時に、ポリシー解決メディエーション・プリミティブは、メディエーション・ポリシー情報をレジストリーから取得します。

メッセージ・サービス・クライアント

C/C++ および .NET 用のメッセージ・サービス・クライアントを利用して、Java 以外のアプリケーションからエンタープライズ・サービス・バスに接続できます。

Message Service Clients for C/C++ and .NET は、Java Message Service (JMS) API と同じインターフェースのセットを持つ XMS という API を提供します。Message Service Client for C/C++ には 2 つの XMS 実装があり、一方は C アプリケーション用、もう一方は C++ アプリケーション用です。Message Service Client for .NET には、完全に管理された XMS 実装があり、これは任意の .NET 対応言語で使用できます。

Message Service Clients for .NET は http://www-01.ibm.com/support/docview.wss?rs=0&q1=IA9H&uid=swg24011756&loc=en_US&cs=utf-8&cc=us&lang=en から入手できます。

Message Service Clients for C/C++ は、http://www-01.ibm.com/support/docview.wss?rs=0&q1=ia94&uid=swg24007092&loc=en_US&cs=utf-8&cc=us&lang=en から入手できます。

また、WebSphere Application Server から、Web サービス・クライアント、EJB クライアント、JMS クライアントを含む、Java EE クライアント・サポートをインストールして使用することができます。

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21