IBM's WebSphere Service Registry and Repository - Technical Overview

Barbara McKee, Marc-Thomas Schmidt, John Colgrave, Duncan Clark

Version 0.7, July 2006

This paper describes the main concepts and capabilities of the WebSphere Service Registry and Repository (*WSRR*). We will explain the role of a service registry and repository in a *Service Oriented Architecture* (*SOA*) and describe the capabilities of IBM's WSRR; we will discuss its relationship with other parts of the IBM SOA Foundation and describe integration with other repositories of *service metadata artefacts*.

Introduction

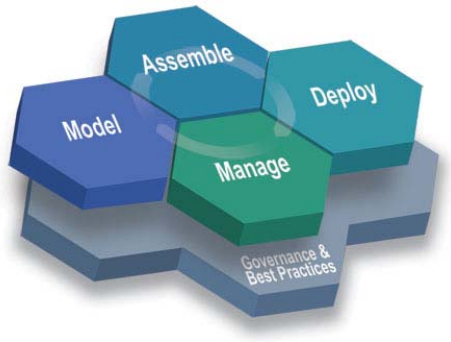
Service Oriented Architectures offer the promise of business agility and resilience through reuse, loose coupling, flexibility, interoperability, integration and *governance*. These are realized by separating *service description*s from their implementations, and using this *descriptive metadata* across the service life-cycle. Standards-based service metadata artefacts, such as *WSDL*, *XML Schema*, *WS-Policy* or SCA *document*s, capture the *technical details* of what a service can do, how it can be invoked or what it expects other services to do. *Semantic annotation*s and other metadata can be associated with these artefacts to offer insight to potential users of the service on how and when it can be used, and what purposes it serves. This service metadata is used by Analysts, Architects, Integrators, and Developers during the *Model* and *Assemble* phases of the SOA life-cycle[1] to locate services and policies to (re)use and to evaluate impact of changes to service configurations; it is used by Deployers and Administrators in the *Deploy* phase of the SOA life-cycle, and exploited by the SOA Foundation runtimes for dynamic selection of service *endpoint*s and configuration of the SOA environment; and in the *Manage* phase of the life-cycle to support policy enforcement required by Service Level Agreements (SLAs) and to present a more comprehensive view of the managed service environment.

Service metadata artefacts exist across an enterprise in a variety of heterogeneous development and runtime stores which often provide information about a service tailored towards use cases in a particular phase of the SOA life-cycle; examples include *Asset Management systems* in the development space or Configuration Management systems in runtime space. In this context, WSRR handles the metadata management aspects of operational services and provides the system of record of these metadata artefacts – the place where anybody looking for a catalogue of all services deployed in or used by the enterprise would go first. It provides ***Registry*** functions supporting publication of metadata about services, their capabilities, requirements and semantics of services that enable service consumers to find services or to analyse their relationships. And it provides ***Repository*** functions to store, manage and version service metadata. It also supports ***Governance*** of service definitions: to control access to service metadata; to model life-cycle of service artefacts; to manage promotion of services through phases of their life-cycle in various deployment environments; to perform impact analysis and to socialize changes to the governed service metadata.

[1] For a detailed description of IBM's SOA Foundation, the SOA life-cycle and the Reference Architecture see http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-whitepaper.pdf

A Day in the Life – WSRR in the SOA Life-cycle

The WebSphere Service Registry and Repository (WSRR) is the master metadata repository for *service descriptions.* We use a broad definition of "service" here, including traditional *Web service*s implementing WSDL interfaces with *SOAP*/HTTP bindings as well as a broad range of SOA services that can be described using WSDL, *XSD* and WS-Policy decorations, but might use a range of protocols and be implemented according to a variety of programming models[2]. As the integration point for service metadata, the WSRR establishes a central point for finding and managing service metadata acquired from a number of sources, including service application deployments and other service metadata & endpoint registries and repositories, such as *UDDI*. It is where service metadata that is scattered across an enterprise is brought together to provide a single, comprehensive description of a service. Once that happens, visibility is controlled, versions are managed, proposed changes are analyzed and communicated, usage is monitored and other parts of the SOA foundation can access service metadata with the confidence that they have found the copy of record. WSRR focuses on a minimalist set of metadata describing capabilities, requirements and semantics of services. It interacts and federates with other metadata stores that support specific phases of the SOA life-cycle and capture more detailed information about services relevant in those life-cycle phases; examples of specialized repositories include a reusable asset manager in development or configuration management database in service management. For the minimalist metadata set WSRR allows users to manage their life-cycle from development through deployment to their use by SOA runtimes and in service management.



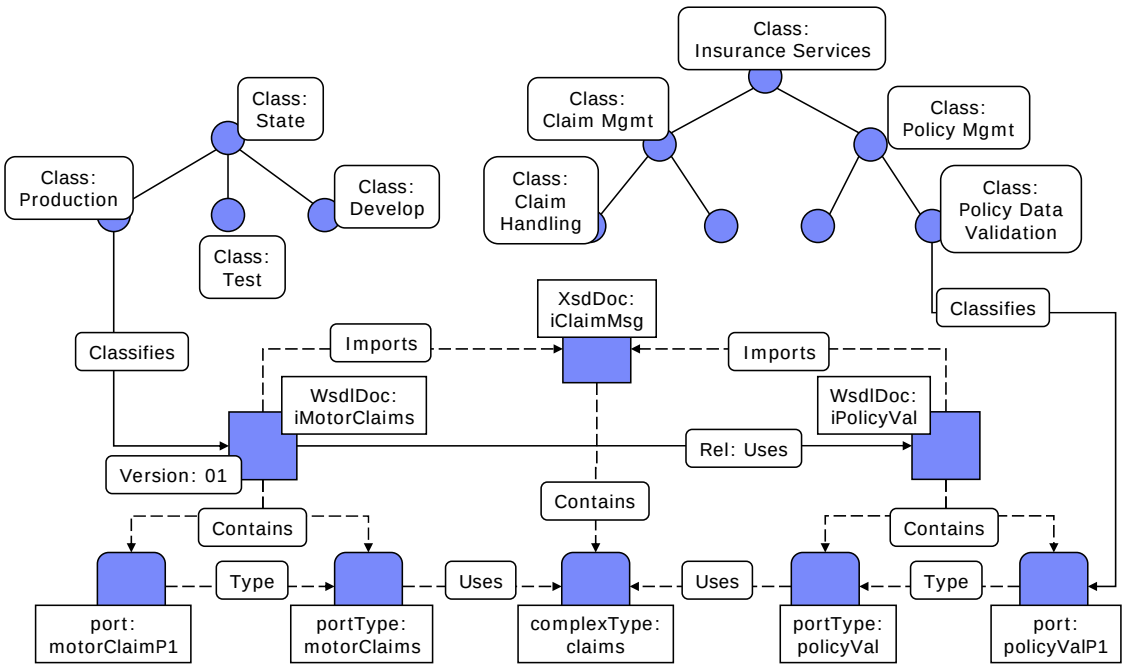
To illustrate use of the WSRR across the SOA life-cycle phases, let's take a look at a day in the life of a business service – from it's invention in the *Model* phase of the SOA life-cycle to its translation into an executable composite application in the *Assemble* phase to its transition into the SOA runtime environment in the *Deploy* phase to it being observed and controlled in the *Manage* phase of that life-cycle.

We will use a simple insurance claims processing example to illustrate the concepts: our insurance company has an existing Motor Insurance Claims process that uses a Policy Validation service; and it wants to expand into Home Insurance and introduce a corresponding Claims handling process. Let's assume that the metadata describing the existing Motor Insurance Claims service and the Policy Validation service it depends on (e.g., the iMotorClaim.wsdl & iPolicyVal.wsdl describing their interfaces and a iClaimMsg.xsd describing the schema of messages they exchange) have been published in WSRR; as a side-effect of that publication, the relationships between the WSDL files and the XSD have been represented in WSRR and that the services have been assigned life-cycle state "production"[3]. Let's further assume that the fact that iMotorClaims uses iPolicyVal has been modelled in WSRR via a "uses" relationship between the two service definitions and that an Insurance Services Classification taxonomy has been imported into WSRR that is used to classify the two services as /ClaimMgmt/ClaimHandling and /PolicyMgmt/PolicyDataValidation respectively.

[2] For more information see "Introduction to the IBM SOA programming model", by D. F. Ferguson, M. Stockton at http://www.ibm.com/developerworks/webservices/library/ws-soa-progmodel/

[3] Often different WSRR repositories will be used to manage service metadata in production versus pre-production; but it is possible, as illustrated here, to partition a single repository to manage both

Figure 1: Simplified WSRR content example



Model & Assemble

During the development phases of the service life-cycle WSRR is used to locate the copies of record of candidate service interaction metadata or intermediaries as well as policies governing the interactions; and it can be used to publish and govern service metadata about emerging, to-be-deployed services.

As in the other SOA life-cycle phases, WSRR is complemented by repositories that specialize in managing SOA artefacts during the development phases of the life-cycle. For example, a development artefact management system such as Rational Clearcase can take care of service and composite application building blocks such as source code, service interface declarations, software architecture models or business process models that are under construction. Asset management systems can manage “bundles” of artefacts describing assets according to the Reusable Asset Specification (*RAS*)[4] standard; it can implement governance processes controlling the promotion of artefacts to assets and the approval process associated with that.

The WSRR complements these repositories and federates data they manage; for example, a given WSDL document in the WSRR can be annotated with a user-defined relationship to a reusable asset declaration that WSRR users can follow to find more information about intended usage and function of that service, or Reusable Asset managers can use to analyze usage of the asset (e.g., how many deployed services are based on it).

[4] See http://www.omg.org/cgi-bin/doc?formal/2005-11-02

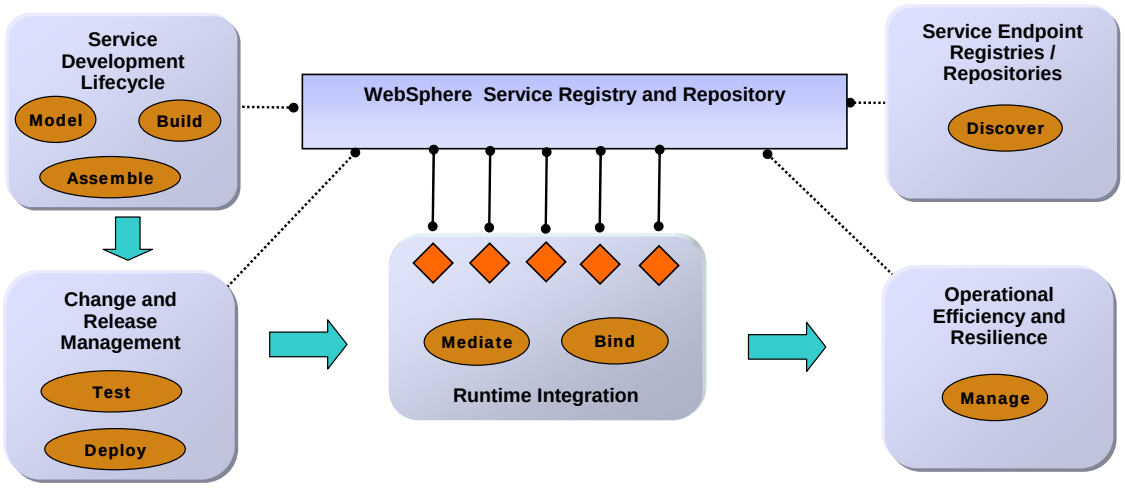


Figure 2: Service Registry & Repository and SOA Tasks

Business Analysts, Solution Architects Component Developers and *Integration Developers*[5] create service metadata and make use of existing service metadata as they perform their specific tasks. They use development artefact management systems to take care of their intermediate work products and contribute to the definition of reusable assets hosted by a RAS manager; they use WSRR to explore the set of already deployed services as building blocks for the new things they are building and they produce service metadata that is published to WSRR once the underlying service is considered ready for being shared in a governed way; classifications associated with service metadata can be used to indicate the life-cycle state of the service (e.g., under construction, in test, ready for production). Or they can use WSRR to understand impact changes they intend to make on existing services would have on other service building blocks. Their interest in and contribution to WSRR content differs according to their specific tasks; here's a brief summary:

- *Business Analysts* model new business processes and can browse WSRR to better understand the existing environment. They are less interested in IT-level technical details of services like their WSDL interfaces, than in a business view on service capabilities established by the WSRR via semantic annotation of the underlying IT-level artefacts. They may work with the *Asset Manager* to define and apply *classifier systems* used to describe business semantics of services. And they may create WSRR entries representing business-level concepts like main applications and processes that will later on be refined by architects and developers into IT-level artefacts.

In our example, an Analyst (see (1) in figure below) could identify the need for a new Home Insurance Claims handling process service and sketch out important process elements such as distinguishing between low and high risk claims. They could use WSRR to discover the existing Policy Validation service that can be used to implement the new process and indicate that a new High Risk Validation service is needed to deal with special cases. They could create a WSRR Concept entity to represent the envisioned new Home Insurance claims business service, relate it to the existing service and use the Insurance Services taxonomy to indicate semantics of the new Concept. They could declare the new entity to be governed and select the life-cycle stage to be applied to it.

- *Solution Architects* use the Service Registry & Repository to find existing services they can use as building blocks for assembly of new composite services; they may also use

[5] For more information about the User Roles used here see "SOA user roles" by M. Chessell, B. Schmidt-Wesche at http://www.ibm.com/developerworks/webservices/library/ws-soa-progmodel10/

service descriptions they find in WSRR to populate the set of building blocks analysts use when modelling new business processes, abstracting the IT specifics into process modelling artefacts. They can publish service metadata that describe interfaces of to be deployed services that will be realized by *Component*- and *Integration-Developers*.

In our example, a Solution Architect (see (2) in figure below) could design the main service components used to implement the new Home Insurance processes; they'd define service interfaces for the High Risk Policy validation service and define the Policy for distinguishing high-risk from low-risk home insurance claims. They'd create WSRR entries for the iRiskyPolicyVal.wsdl and the iRiskyPolicy.policy documents and relate them to the Home InsurancecClaims Concepts created by Analysts and the existing iPolicyVal and iClaimMsg.xsd metadata. They might also request a new version of the Policy Validation service to be built to reflect additional requirements in the new process and use WSRR impact analysis to determine which applications would be affected by a change of this service.

- *Component Developers* build new service components and can browse the WSRR to find services they could invoke as part of their implementation or to scope queries against WSRR they want to use as part of their service implementation. They can use the WSRR to analyse the impact an envisioned change on a service might have on other services; and they can publish service interfaces and other service metadata to WSRR that need to be put under Governance control.

In our example, a Component Developer (see (3) in figure below)would take on the task to implement the new High Risk Claims handling service based on the iRiskyPolicyVal.wsdl service interface specification provided by the Solution Architect. They could attach a reference to the actual service implementation artefacts in a source code library to the WSRR artefact.

- *Integration Developers* assemble solutions from new or existing components. They use WSRR to find building blocks they can bind references in their composite applications to; and they model the mediations necessary to facilitate interactions between service interaction endpoints, e.g., using WSRR lookup to select a service endpoint to be used in a dynamic routing scenario. In coordination with the *Asset Manager* role they also can discover existing service endpoints (e.g., external service offered by a business partner) and publish relevant service metadata in the WSRR.

In our example, the Integration Developer (see (4) in figure below) could implement an SCA Mediation Integration Module that handles routing of Home Insurance Claims to either the regular or the high risk Policy Validation service; the module would implement the iHomeClaims interface and declare a dependency on a service implementing the iPolicyVal interface. They would implement mediations that lookup potential endpoints in WSRR and pick the one that has a riskyPolicy.policy attached to it.

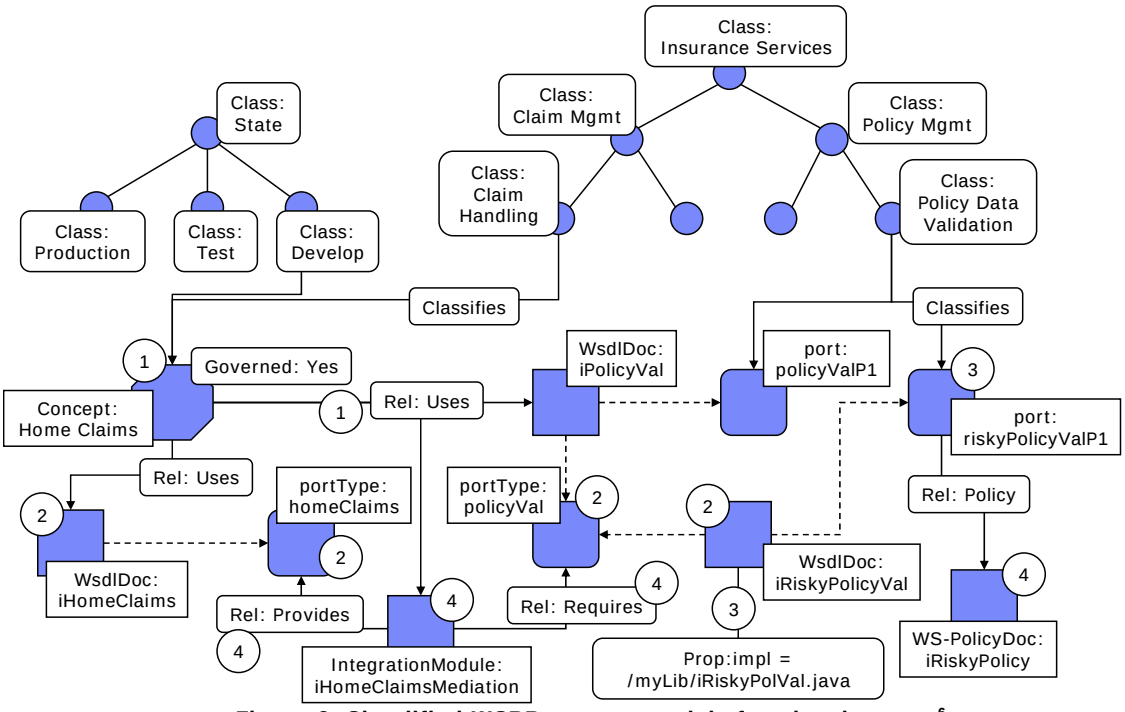


Figure 3: Simplified WSRR content model after development[6]

Deploy

The WSRR provides the system of record for metadata describing service interaction endpoints. It is populated with metadata as part of SOA solution deployment or via discovery of existing endpoints; and it is used by SOA runtimes as well as *Deployer* and *Administrator* roles when detailed information about service endpoints is required to drive operations of the deployed composite applications.

Once the development team has finished their work and testing is complete, *Deployers* further augment the service metadata, providing binding information for service endpoints used in composite applications and managing deployment of the metadata from the development environment to the staging or production instance of WSRR as part of the service deployment process. Governance over the service metadata takes place, as metadata is promoted from test to staging to production environments that may have separate WSRR installations. In the production WSRR, service metadata is made available to a broader audience and shared. It is available to the runtime systems and those user roles that are responsible for the management of the IT systems.

In our example, a Deployer (see (1) in the figure below) would promote the Home Insurance Claims business service from test into production. Most importantly, they would update the service bindings for the new service components to reflect their deployment into a production environment; they would update the endpoint declaration in the port definition of the iRiskyPolicyVal service and add a service binding to the iHomeClaimsMediation interface. They would transition the state of the application to (a sub-state of) the production state classifier – if the validations associated with this life-cycle state transition are successful. If multiple WSRR repositories are used, the promotion would include export of service metadata from the pre-production WSRR and import of the updated service endpoints into the production WSRR.

Here again, service metadata can be discovered in other service registries and

[6] The SCA portion of the model is significantly simplified here; alternative realizations of the scenario without use of SCA Mediation module are possible

repositories, published in the WSRR, and can be used as input for the application configuration and binding tasks performed by the Deployers and Solution Administrators. Discovered service metadata is usually incomplete and not yet suitable for broader visibility and consumption. Deployers work with *Asset Managers* to ensure the metadata is augmented with necessary semantics, permissions and scoping constraints.

Automation of service metadata publication during service application deployment integrates the management of service metadata with the overall SOA management and governance processes in a first class way.

Execution time interactions with the WSRR can be implemented in a service endpoint by a *Component Developer* or by a mediating intermediary which acts on behalf of the service requester or the provider and is configured by an *Integration Developer*. Dynamic endpoint selection is a key use case that involves querying the WSRR for the service provider or a set of candidate service providers, and binding to the endpoint that is the best choice. Endpoint selection rules can be encoded in ESB-managed mediations[7].

In our example, the HomeClaimsMediation service would perform a lookup (see (2) in the figure below) of the Policy Validation service to be used; it queries for service endpoints (i.e., ports) that implement the PolicyVal portType and are classified as Policy Validation services; depending on the content of the request to be processed it will then pick either a vanilla one, or, if a high risk claim is to be handled one that has a relationship to the iRiskyPolicy WS-Policy declaration.

Another use case is dynamic retrieval and enforcement of the policies that are in effect for a service interaction in the areas of logging, filtering, data transformation, or routing. WSRR can store the policy definitions; service interaction endpoints can reference applicable policies; policy enforcement points, such as mediations, can be configured using that information and can be updated when the WSRR-managed metadata changes.

SOA runtime elements will often maintain configuration information beyond the minimalist set of service metadata managed by the WSRR; they will cache WSRR managed metadata and relate it to the additional configuration information they maintain.

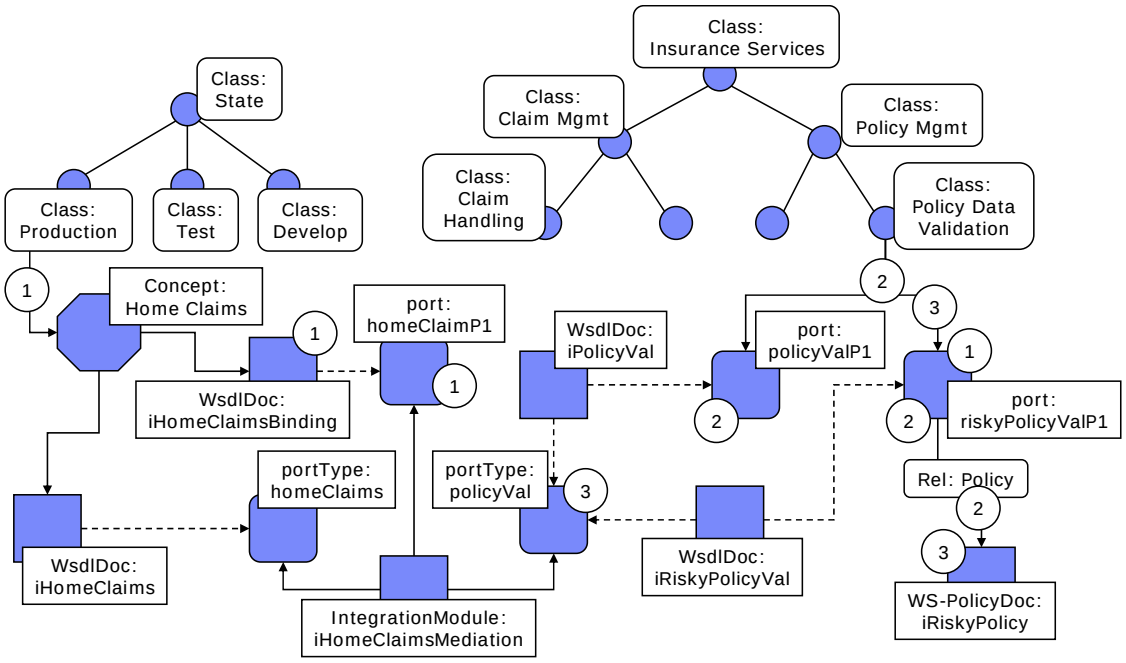
Solution Administrators use WSRR content to better understand the solution artefacts they are administering, and in the future may be able to dynamically affect changes in the configuration of deployed applications by updating WSRR content (e.g., replacing documents describing endpoint selection rules or applicable policies).

In our example, a Solution Administrator (3) might observe increased traffic between the homeClaimP1 and the riskyPolicyValP1 endpoints and use WSRR information to understand the nature of the services involved (e.g., retrieving information about portTypes or Classification of the service endpoints) and the relationship between those services; they could modify the riskyPolicy WS-Policy definition to adjust routing criteria in the iHomeClaimsMediation.

Clearly those types of administrator-driven updates must be controlled and that's achieved using the WSRR governance support for the artefacts in question which controls visibility of artefacts as well as who can perform which actions on specific governed entities.

Figure 4: Simplified view on WSRR content in deploy phase

[7] For more details see "An introduction to the IBM Enterprise Service Bus", by B. Hutchison, M-T. Schmidt, D. Wolfson, M. Stockton at http://www.ibm.com/developerworks/webservices/library/ws-soa-progmodel4/



Manage

Operational management and resilience within the SOA is enhanced by sharing the service metadata that exists in the WSRR with operational data stores, allowing management and monitoring dashboards to present a more comprehensive view of the managed service environment. Summary information about service performance can be fed back into the WSRR and used by the execution environment to affect the selection of the best-fit provider.

As in the other life-cycle phases, WSRR only manages a minimalist set of service metadata and federates with repositories specializing on managing information about services in this life-cycle phase. For example a Configuration Management Database (*CMDB*) like Tivoli's Change and Configuration Management Database will acquire and manage detailed information about the environment & topology in which service endpoints execute; and service management products like IBM IT Service Management suite will use and update that information and drive governance of processes that provision and configure the underlying infrastructure.

In our example, a CMDB would manage information about the topology into which the Home Claims service and its building blocks are deployed; for the homeClaimsP1 port it would have information about the environment it is installed into. A reference to the CMDB entry that allows navigation to this information could be established in WSRR (see (1) in the figure below). IT service monitoring applications such as ITCAM could provide summary health information about service endpoints in WSRR and trigger updates of a service health reporting property on those endpoints in WSRR[8].

WSRR and CMDB federation enables WSRR users to drill down into information about environment and runtime status of a service while CMDB exploiters can obtain detailed descriptions of syntax and semantics of service endpoints from WSRR. Service monitoring and management products like ITCAM for SOA provide instrumentation of service interactions that allow monitoring of service interactions and service endpoint

[8] WSRR would only provide summary information about service status (e.g., whether it is active or not) , not in any way real-time status information (e.g., current throughput)

behaviour; summary information about service behaviour can be pushed into WSRR to decorate service endpoint metadata with execution statistics and that information can in turn be used by *Integration Developers* or SOA runtimes to understand state or usage of a deployed service. In the future service management products can use WSRR managed policy information to configure policy enforcement points that implement the SLAs users want to enforce in service interactions.

In the *Operator* user role category, the *Incident Analyst* investigates unexpected incidents when they occur in the IT system and manages service endpoints. They consult the metadata about service endpoints found in the WSRR to understand the behaviour of the IT systems and to assess the impact of an underlying failure; they retrieve information to augment what is known about services from monitoring their interactions and can obtain information about the life-cycle state of a deployed service. And the *Business Operations Manager* analyses performance of SOA applications from a business value perspective and uses metadata provided from the WSRR to understand application semantics and to compare expected with observed behaviour of those applications.

In our example, an Incident Analyst (see (2) in figure below) could observe a problem with the iPolicyVal service endpoint and use WSRR to understand that any homeClaims and motorClaims service could be affected by that problem.

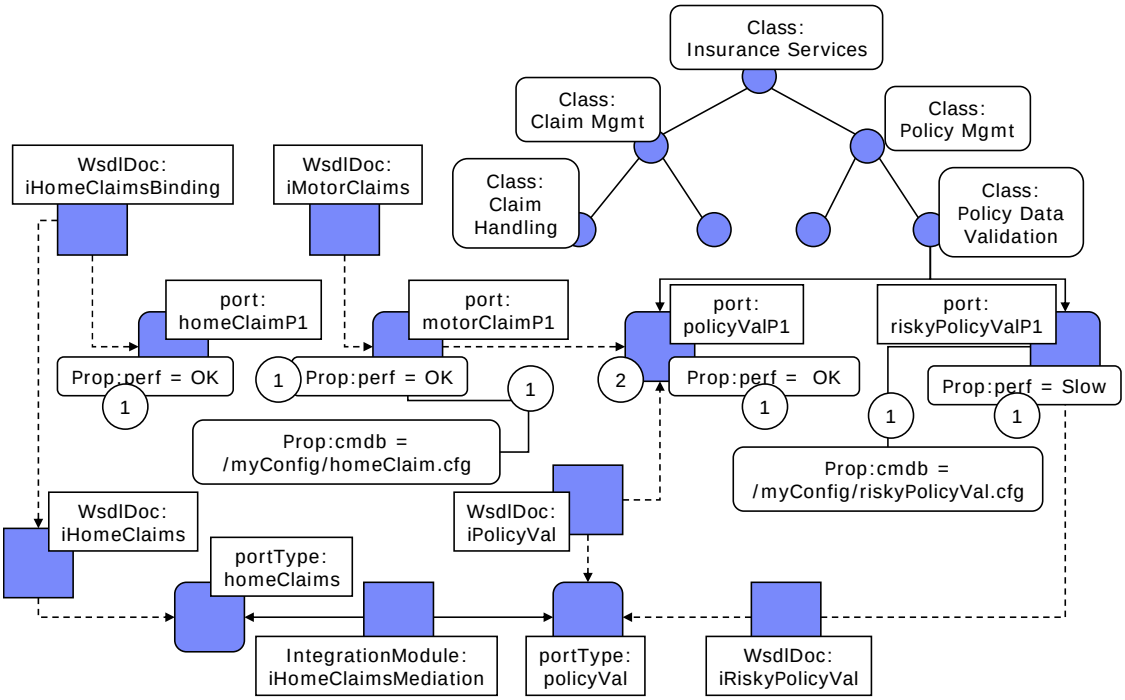
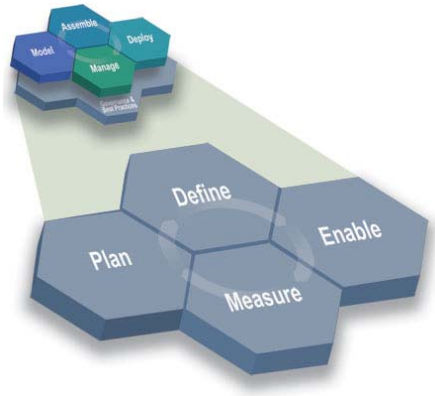


Figure 5: Simplified view on WSRR content in Manage phase

Governance

The WSRR plays a key role in the end-to-end Governance underpinnings of the SOA life-cycle[9]. Simply put WSRR support governance of service metadata throughout the life cycle of a service from its initial publication in development space to deployment to service management.

WSRR is mostly used in the *Enable* phase of the Governance cycle where Governance policies and processes are actually implemented, for example to implement service life-cycle models or metadata validation rules or to define access control for service metadata; as well as in the *Measure* phase where governance requirements are monitored and measured and WSRR information enables the monitoring process. It is also used in the *Plan* phase where governance needs are defined, e.g., to review existing governance capabilities realized in WSRR or to define the basic life-cycle model for governed entities. And it is used in the *Define* phase where the Governance approach is specified to, for example, capture details of the life-cycle models for service artefacts or SLAs governing service interactions.



The *Enterprise Architect* plays a key role in defining the Governance policies and processes for an enterprise. This user role defines the process that the IT organization will use to develop, run and maintain services. This process is reflected in the setup of the WSRR and is reviewed and improved by the Enterprise Architect over time. For example, governance policies and processes describe the rules for staged rollout of new services from development to staging to production; determine who can access or modify which service artefacts at what phase in their life-cycle; define validation rules for services to be considered valid in a specific life-cycle phase.

In our example, an Enterprise Architect could define the basic life-cycle model for services in our insurance company – this could be as simple as having three states: develop, test, production (as in the illustrations above) or it could have more sophisticated sub-states for the various development phases. They would also define access policies for user roles performing actions on services; in general those would be generic rules such as "developers cannot promote services from test to production" or "only developers have visibility to services with state classification "develop". And they would identify the Classification system to be used; in addition to a state model they would pick the Insurance Services taxonomy.

The *Asset Manager* plays a key role in translating governance policies and processes into production. They are responsible for the correct use and management of IT assets, including the service metadata in the WSRR. They use WSRR governance features in combination with governance features provided by other and more SOA life-cycle phase-focussed tools (such as a reusable asset manager in development space or Service Management products driving governed service provisioning and management processes), to make the SOA Governance processes operational. Governance actions that the Asset Manager role performs on service metadata are:

- driving discovery and publication of new services in cooperation with Analyst, Architect, Developer and Administrator roles often in the context of managing reusable assets;
- driving definition of life-cycle models for metadata associated with deployed (or soon-to-be-deployed) services;
- driving implementation of validation rules against WSRR content to maintain the integrity of the service metadata;

[9] For more information about SOA Governance see http://www-306.ibm.com/software/solutions/soa/gov/

- driving implementation of rules for who can affect which service metadata life-cycle changes under which conditions and with which side-effects;
- driving configuration of WSRR installations to support a staged rollout of services and their metadata from a test to a staging to a production WSRR;
- assigning *classifier*s to service descriptions so they can be located and used;
- selecting categorization systems to use to annotate service metadata artefacts;
- identifying and captures version information for service metadata;
- configuring WSRR to enable sharing and integration with other metadata and asset stores;
- defining access rights for WSRR users to enforce service visibility and promotion rules;
- defining rules for propagating notifications of changes to WSRR managed service metadata and
- supporting analysis of upcoming changes, communicating such changes, and realizing those changes that are authorized.

In our example, Asset Managers would load the Insurance Service taxonomy into WSRR; they would also provide a more detailed classification of the building blocks of the homeClaims service and provide implementations for the validation rules to be applied; again, most of those rules are probably service endpoint independent, so there is not much specific work to be done for our example here. Finally, since Asset Managers are responsible for the integrity and visibility of the service metadata throughout the organization, they would transition the service metadata through its more advanced and visible life cycle states, such as the transition from pre-production to production.

WebSphere Service Registry and Repository Architecture

In this section we discuss the WSRR architecture and capabilities that support the use cases described above: the content model of the repository, registry and governance capabilities, the interfaces for interacting with the WSRR, and the federation model for integrating with other service metadata stores.

Overview

A servlet-based Web User Interface is the main way for users representing different roles to interact with the WSRR, supporting lookup and publish scenarios, metadata management and analysis scenarios, and functions that support SOA governance. An Eclipse plug-in is also provided to support lookup, retrieval and publishing of service metadata in the context of eclipse-based development tools or management consoles.

Programmatic interactions with the WSRR are provided through both Java APIs and SOAP APIs. Basic CRUD operations as well as governance operations and a flexible query capability based on XPath are provided through both APIs. When the SOAP API is used, content is communicated using XML data structures; when the Java API is used, content is communicated using SDO data graphs.

The WSRR core offers both Registry and Repository capabilities. The repository allows users to store, manage and query content of documents holding service metadata descriptions (WSDL, XSD, WS-Policy, *SCDL* or XML documents). It not only takes good care of the documents containing service metadata but it also provides a fine-grained representation of the content of those documents (e.g., ports, portTypes in WSDL documents). And it provides registry functions for decorating registered service declarations and elements of the derived content models with *user-defined properties*, *relationships* and *classifiers*, and a rich query interface that makes use of those decorations when users want to find a service endpoint, interface description or other metadata about a service.

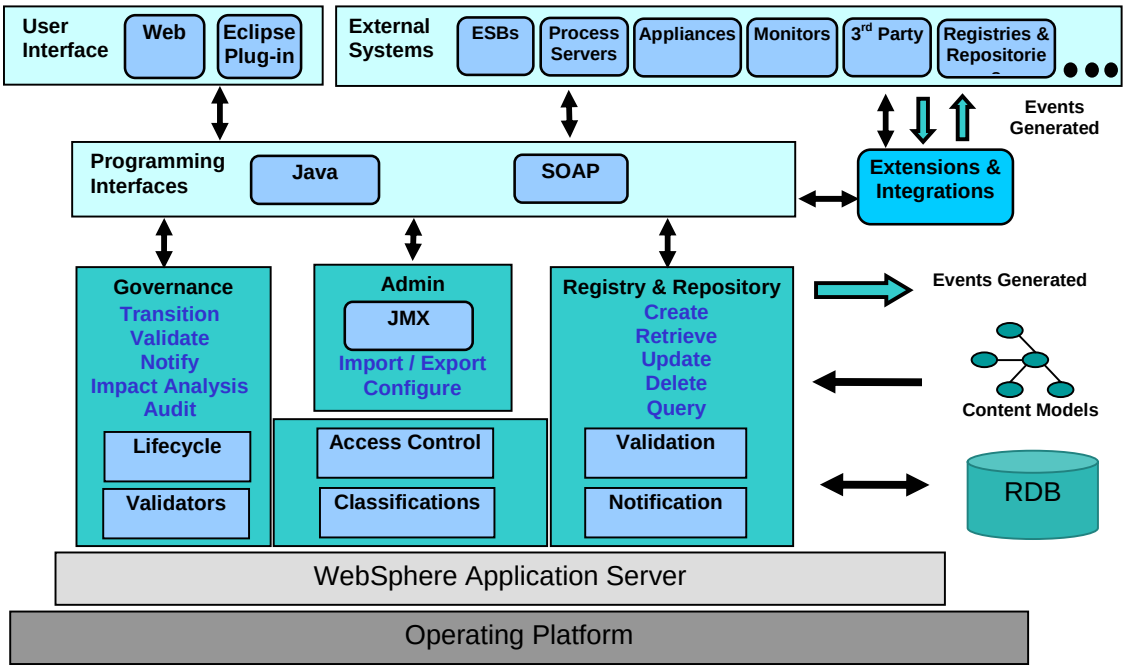


Figure 6: WebSphere Service Registry and Repository Architecture

WSRR allows users to plug-in validation functions to be executed when changes are made to the repository content (e.g., checks for completeness of a service definition). It also provides notifications of any changes to the content of the repository and allows users to register their interest in consuming those notifications.

The WSRR supports a fine grained access control model that allows users to define which user roles can perform what kind of actions on which artefacts. It allows users to define and import Classifier systems from simple classifier sets to taxonomies and classification hierarchies.

The WSRR supports a rich set of Governance functions, including a life-cycle model for governed entities using a state machine that describes these states, valid *transitions* between them, plug-invalidators to guard the transition and (notification) actions to be taken as result of the transition. It also provides interfaces to analyze impact of changes to WSRR content and auditing of such changes.

The WSRR administration interfaces support import and export of WSRR content for exchange with other metadata repositories (e.g., other WSRR installations) and provide a JMX-based API for WSRR configuration and basic administration.

The WSRR is a J2EE application based on the WebSphere Application Server (WAS) and uses a relational data base as a backing store for service metadata (e.g., DB2, Oracle) via the xMeta metadata management framework used in many IBM products that deal with XML metadata (e.g., the products in the WebSphere Information Server suite).

Content Model

In the following paragraphs we provide a rather informal description of the WSRR content model; a more formal and complete description is provided in the WSRR Programming Model [WSRR Programming Model]: in a nutshell the content model has entities representing service description artefacts and service description metadata. All WSRR content elements have a WSRR-assigned URI, a name and a description.

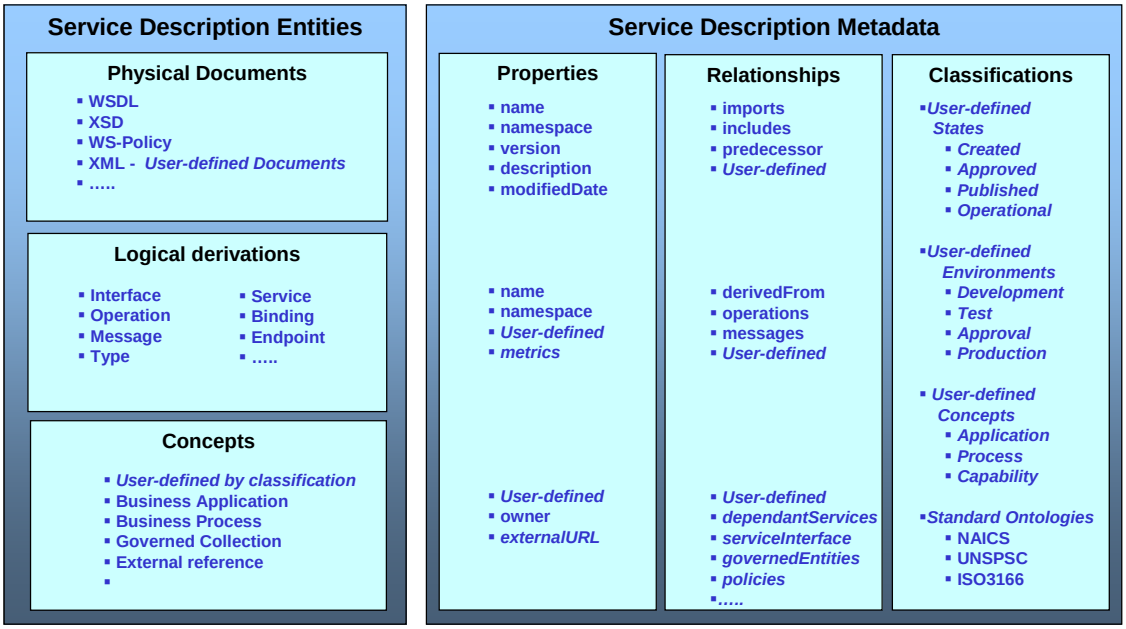


Figure 7: WSRR Content

Let's start with the most elemental building blocks for the WSRR content model: service metadata artefact documents (Physical Documents), such as XSD or WSDL files. These service metadata documents are stored and managed in the WSRR; we refer to the coarse-grained model made up from registry objects that represent those documents as the *physical model*; documents are versionable objects in the WSRR content model which basically means that in addition to a URI, name and description they also have a version property.

Any *service metadata artefact type* can be stored in the WSRR and receive the benefits of broader visibility and reuse, management, and governance. However, WSRR offers advanced functions for a number of well-known SOA metadata types. The key WSRR metadata document types are: WSDL, XSD, WS-Policy and SCDL; for those document types the WSRR provides special services, including "shredding" of the documents upon receipt into Logical Derivations or a set of Logical Objects to enable users to explore WSRR content beyond the boundaries of the files stored. Logical objects cannot be individually versioned since they are derived from a physical document (which can be versioned) and cannot be individually manipulated.

For the key document types WSRR also defines a few pre-defined properties and makes an effort to detect relationships to other key documents, and where available, records those relationships in the content model. An XSDDocument, for example, has a targetNamespace property and the relationships importedXSDs, redefinedXSDs and includedXSDs. When an entry for a key-type document is created in the WSRR, it is introspected for relationships to other key-type artefacts; if not already in represented in WSRR, these related artefacts are also added, and in either case the relationship between the artefacts is recorded.

The set of logical derivations comprises the *logical model* of the WSRR. The logical model has entities such as portType, port, and message related to WSDL files, and complexType or simpleType related to XSD documents. Elements of the logical model have properties and relationships reflecting a subset of their characteristics as defined in the underlying document. For example, a WSDLService element has a namespace property and a relationship to the ports it contains. It is important to note that all individual results of document shredding are aggregated into *one* logical model that represents not only the content of individual documents, but also relationships between content in different documents.

Other types of service metadata can be stored using a generic content type: XML Document; documents of type XMLDocument are not decomposed into the logical model.

There is one other kind of *entity* in the WSRR, loosely referred to as a *Concept* in the figure above. This is a *generic object* that can be used to represent anything that does not have a physical document in the WSRR. Concepts can be used to represent a reference to content in some other metadata repository such as a portlet in a portlet catalogue, an asset in an asset repository, service implementation artefacts kept in a source code library or information about SOA infrastructure topologies in a configuration management database. It can also be used to group *physical artefact*s together for ease of retrieval, for example. Concepts can be versioned. WSRR provides a set of pre-defined Concepts that offer a business-level view on the SOA metadata managed in WSRR; this basic SOA model includes concepts such as Business Process, Business Service, business Object and Business Policy.

In addition to content directly related to service metadata documents, the WSRR supports a number of user-defined metadata types that are used to decorate the service metadata to explain their semantics; we refer to those metadata as Service Description Metadata.

WSRR supports three types of service semantics metadata types: Properties, Relationships and Classifiers; all three types can be used to decorate entities in the physical or logical model, and concepts as well. Users can associate a property “businessValue” with a physical model entity representing a WSDL file; or they can define a new relationship “makesUseOf” between an entity in the logical model representing a “portType” and an entity in the physical model representing an XML document; or they can create a classifier “importantThings” and associate it with a “port” entity in the logical model and with an entity in the physical model representing a “Policy” document. This enables semantic queries to target individual elements of the service metadata, and meaningful dependency analyses to take place prior to making changes.

User-defined category systems are imported and shared through the use of documents encoded using the *Web Ontology Language* (*OWL*)[10]. While any valid OWL document can be used as a Classifier system, at this point in time WSRR exploits only a small subset of the expressiveness of OWL – it represents OWL Classes as classifiers and interprets the subTypeOf relationship between those Classes as establishing a classifier hierarchy; other OWL concepts such as data or relationship representing properties or other built-in OWL relationships are ignored.

A classifier system is imported into WSRR as a whole and can not be modified via WSRR functions; updates are done by importing a modified version of the *ontology*. Any Class in the underlying ontology can be used as a classifier; the same classifier can be used to classify multiple entities and an entity can be associated with multiple classifiers.

User-defined properties and relationships can be used to customize the set of pre-defined

[10] See http://www.w3.org/2004/OWL/ for details

properties and relationships provided in the WSRR meta-model. Users can add properties to a WSDLDocument or they can configure a concept with properties and relationships to represent its structure; to simplify the process of customizing WSRR entities, a simple "template" mechanism is supported that allows users to identify an XSD complexType as a template for definition of properties and relationships for a customized entity and by associating the template schema with the entity drive creation of the required additions.

Registry and Repository Capabilities

The WSRR supports basic Create, Retrieve, Replace and Delete operations for the documents it manages; content of the service metadata documents or of the logical model derived from them cannot be modified via WSRR interfaces. Users can, however, Create, Update and Delete semantic decorations on those artefacts and on the logical model elements.

The WSRR query interface offers a set of predefined queries that are configured via user provided parameters (example: all WSDL documents with classifier X) as well as generic, XPath based query capabilities. For the Java interface the XPath expression identifies the type of WSRR managed entity to be returned and a filter in terms of WSRR managed elements related to the desired object. Extensions are provided to include classifier annotations in a query. For example, to return all WSDLServices that have a port that refers to a binding that refers to a portType named "StockQuotePortType" the following query expression would be used:

```
//WSDLService[port/binding/portType/@name='StockQuotePortType'];
```

WSRR allows users to register Validation functions to be run when basic CRUD operations are executed against WSRR content and also in the context of the Governance model (more on that below). Validation functions must not have side-effect and must return a Boolean result that can be used to potentially veto the update.

The WSRR provides basic event notification features to allow exploiters to register their interest in any changes to WSRR content. Initially notification will be based on JMS publication of events. WSRR events identify the type of the change and contain a pointer to the object that was changed. In the future we will also support a notification interface based on the emerging WS-* standard in this space.

Both for validation and notification the WSRR product will provide some predefined content; examples may include an e-mail notification feature that would allow users to request an e-mail to be sent when an interesting event happens; or a WS-I compliance checker that can be used in validation scenarios.

The WSRR provides a rich access control model that allows Asset Managers to specify which user role can access which resources and what actions they can perform on those resources. Specification of advanced access control rights is based on the eXtensible Access Control Markup Language (XACML) standard[11].

Governance Model

The WSRR plays an enabler role for SOA Governance, offering increased visibility, with control over reuse, service compliance validation, and change impact analysis. Governance is supported in the WSRR content model through predefined properties and relationships, state machines, and a number of API and UI functions that can be used in governance processes. Life-cycle phase, operational state and service definition status are all used to offer a mechanism for (re)use visibility and operational readiness and

[11] See www.oasis-open.org/committees/xacml for details

health.

At the centre of the WSRR Governance model is the concept of a Governed Entity. Concepts, documents and collections can be governed entities. Users can decide to govern a WSDL document or a collection of documents, for examples a WSDL with related XSDs, a few WS-Policy documents and some XML document that are associated with the WSDL via user defined relationships.

The WSRR features mentioned above such as content versioning, fine grained access control and change notification can be applied in support of service governance.

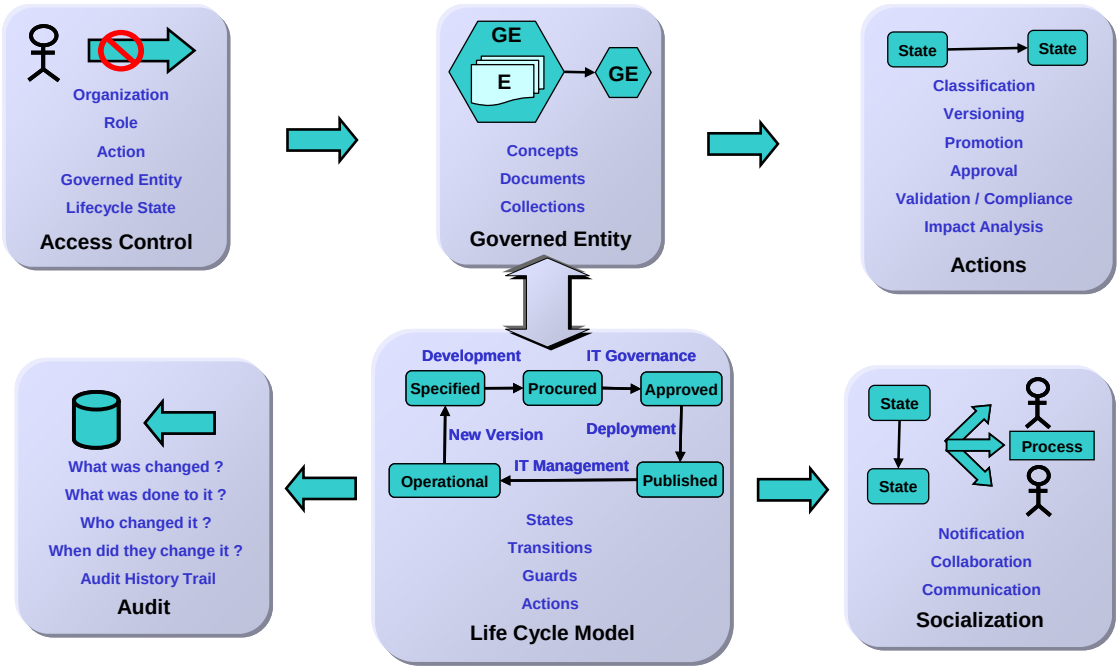


Figure 8: Aspects of the WSRR Governance Model

For each governed entity a state machine is defined that describes the life-cycle states the entity can take on, the transitions between those states, conditions for the transitions to be taken and actions to be taken should a transition be performed. WSRR provides a set of example state machine definitions but Asset managers can customize those or define new ones as required to model the life-cycle of their governed entities. The WSRR governance API allows users to request a transition to be performed on a governed entity; if the entity is not in a state that allows the transition, the request is rejected. Otherwise the conditions for the transitions are checked; these may include access control restrictions prohibiting the requester from performing the transition; or they may specify validators to be successfully executed before the transition can be performed (e.g., check for certain documents to be present in a governed entity collection). If the checks are successful, the transition is performed and associated actions are executed; these may include creation of a notification event reporting on the transition, or updates to the change of the visibility of the governed entity. As an added note, the life-cycle states modelled in the state machine are WSRR Classifiers and can be used in WSRR queries.

The WSRR API includes governance functions such as impact analysis queries and explicit requests for running content validators.

In summary, the capabilities that the WSRR provides in support of governance activities

include:

- The ability to provide OWL-based user-defined semantic classifiers on all parts of the content model, including operations, data types and interfaces to provide consistent and agreed terminology across the enterprise, facilitating the sharing of service metadata
- Control of visibility over and access to service metadata for sharing and reuse
- Support for the tracking of service metadata as it makes its ways through its governed life-cycle, including approvals, deprecation, and retirement, in development, test, staging and production environments
- The ability to perform automatic and upon-request user-defined validation of state transitions
- A dependency analysis function to assist in the assessment of the impact of a change. The WSRR captures many dependencies automatically, as a side effect of storing artefacts in the WSRR
- Change notifications using user-defined notification schemes, basic JMS publication of events providing basic information about what happened as well as an e-mail based notification feature.

Programming Interfaces

We support two flavours of APIs that can be used to interact with the WSRR. One is Java based and the other is SOAP based. Both support publishing (creating, updating) service metadata artefacts and metadata associated with those artefacts, retrieving service metadata artefacts, deleting the artefacts and their metadata, and querying the content of the WSRR.

Clients are provided for both types of API. Service Data Objects (SDO) capture the data graphs inherent in the content model, allowing access to physical documents, logical parts of the physical documents, and concepts. The following discusses the Query and Admin aspects of the APIs in a bit more detail.

The Query API allows the use of XPath expressions to perform unanticipated coarse- and fine-grained queries. Queries can be performed using semantic annotations, properties, and all or parts of physical service metadata artefacts. Fragments of metadata can be returned (such as properties for name or endpoint address), all metadata can be returned (data graph), and metadata and documents can be returned. In addition to “free-form” XPath-based queries, a set of pre-canned queries are offered to address common paths through the WSRR content model.

The Governance API allows users to perform impact analysis of changes to specific artefacts; a set of pre-defined impact queries supports patterns of navigation through the WSRR content such as: which WSDL files import or use this XSD. In addition it provides operations to request life-cycle transitions for a governed entity as well as configuration of e-mail notifications for users interested in specific WSRR content changes.

WSRR also provides a JMX-based Administration API that supports basic configuration as well as loading and managing metadata in support of WSRR content classification and life-cycle management: the API allows users to load definitions of state machines to be used to model the life-cycle of governed entities as well as loading of OWL-described classifier systems. In addition, the Administration API support registration of plug-ins for Validation functions or potentially even addition Notification providers. As explained earlier, Validation functions can be used to control basic CRUD operations as well as in the context of life-cycle state transitions for governed entities.

WSRR ships a default notification handler that publishes change events on a JMS topic; the event specifies the type of event (create, update, delete or transform), the artefact

impacted (identified via its URI) and a few more bits of information about the artefact; to avoid access control problems the actual content of the artefact in question is not shipped with the event but has to be retrieved separately.

A simple Audit capability is provided that basically logs information about WSRR updates to a file.

User Interfaces

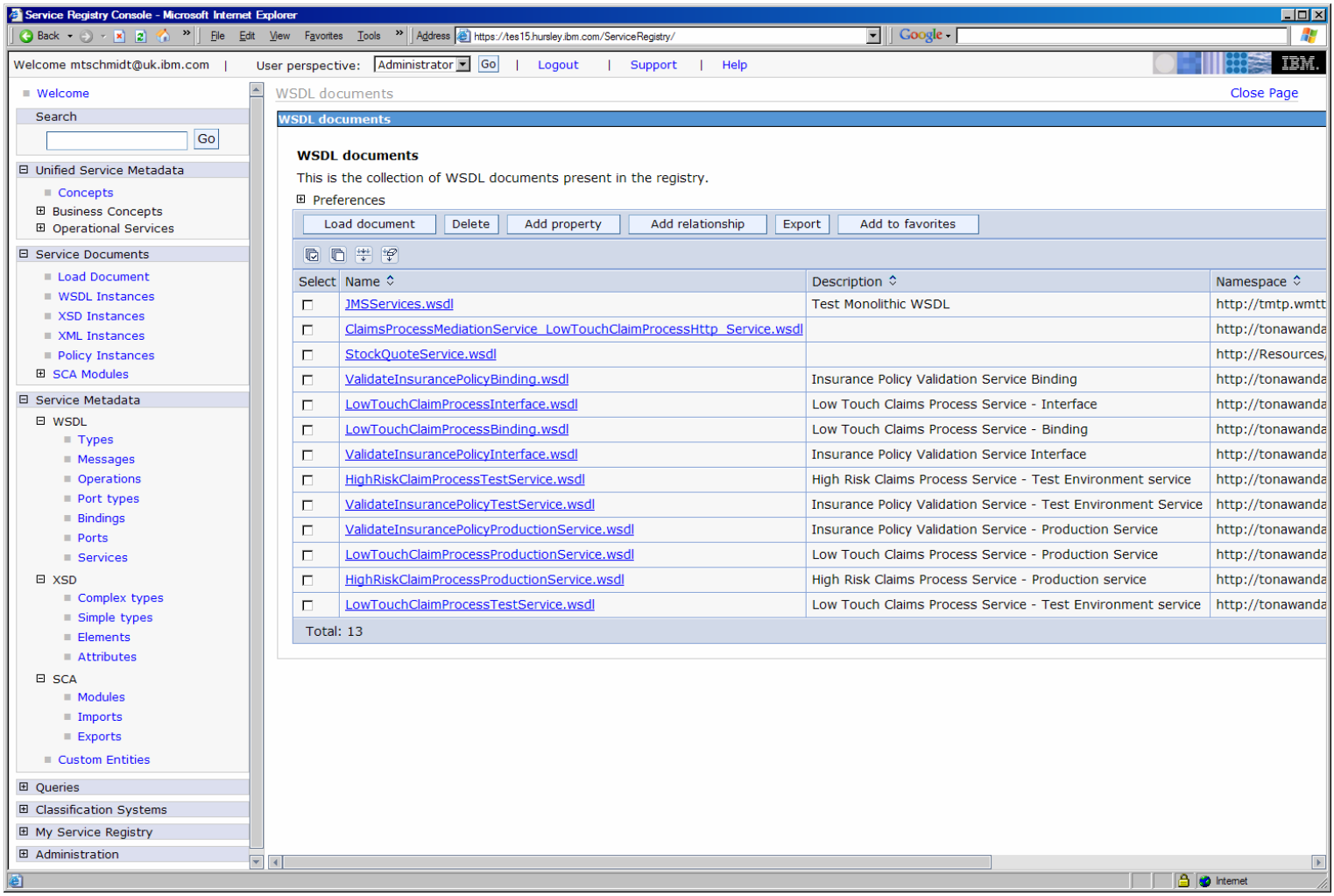


Figure 9: WSRR Web interface snapshot

Two user interfaces are provided to access the WebSphere Service Registry and Repository. The main interface is a Web application deployed with the WSRR runtime. This supports all anticipated user roles, offering lookup, browse, and retrieve capabilities; publish and annotate capabilities; and governance activities, including import/export and impact analysis. A subset of this user interface is offered as an Eclipse plug-in to meet the needs of developer and analyst roles that use Eclipse based-tooling. The Eclipse plug-in is used primarily for lookup, browse, retrieve and publish capabilities. The Web-based User Interface is used for performing service metadata management and governance.

The Web interface supports customization of the views on the WSRR content represented to a user. A set of user interface definition files describes content and layout of the various components that make up the WSRR web interface; the concept of user-role-

specific perspectives is supported. WSRR comes with a set of pre-defined perspectives for the most common user roles but WSRR users can customize the predefined ones or introduce new, role-specific perspectives.

Sources of Service Metadata

Service metadata comes to the WSRR from a number of sources. Developers may be allowed to publish service descriptions to the WSRR to socialize definitions of to-be-deployed services. Very often a developer-provided description will be incomplete and an Asset Manager will add semantic annotations and other metadata to facilitate (re)use and discovery. Service descriptions can also be placed into the WSRR during service or service-oriented application deployment.

Discovery utilities can be written to automatically retrieve service metadata from existing endpoint repositories, from service runtime deployments, and from other service registries and repositories. Discovered service metadata is copied to the WSRR. This enables the WSRR to hold the copy-of-record of service metadata artefacts, where they can be managed and made more broadly visible for reuse. Discovery utilities often capture an external reference to the source of origin to allow the artefacts to be synchronized using notifications, should the artefacts themselves change while residing in the WSRR.

IBM's WSRR will also federate with UDDI registries to the extent that the UDDI registry use is restricted to best practices endorsed by OASIS, covering mapping to the UDDI data model. UDDI federation is based on a copy-and-synchronize approach. WSDL and XML Schema documents that are referenced from UDDI registries will be copied into IBM's WSRR where they will then be available for reuse, analyses and management. Notifications of UDDI changes and additions are subscribed to and used to publish corresponding content into the WSRR. Similarly, relevant changes originating in the WSRR are published to UDDI.

Federation with other metadata registries and repositories that complement the minimalist service metadata managed by the WSRR, such as development asset repositories, IT configuration repositories, business performance repositories, IT governance repositories, and service performance repositories, is enabled through the use of Concept objects in the WSRR. WSRR uses Concepts to represent artefacts in those repositories that are required to establish the link; the foreign artefact Concept provides a URI reference to the artefact, and can be associated with other parts of the service definition. For example, a Concept representing a business asset can be associated with a service. These external reference Concepts are typed, contain a location and identifier, and can be made visible to dependency analyses. Reciprocal references to WSRR content from other metadata repositories is facilitated with the unique id assigned to every service metadata artefact in the WSRR. Real-time synchronization with these other repositories is made possible using the notification API, and manual synchronization through WSRR crawlers is supported with the retention of modification timestamps.

Acknowledgements

Special thanks to the following for their contributions in shaping the WebSphere Service Registry and Repository: Mandy Chessell, Raymond Ellis, Andrew Hately, Beth Hutchison, Ed Kahan, Susan Malaika, Manish Modh, Birgit Schmidt-Wesche, Mike Starkey, Willi Urban, Dan Wolfson, Greg Flurry, Rachel Reinitz, Steve Graham, Michael Ellis.

Glossary

Artefact: In the context of WSRR, a document that contains descriptive information about or relating to a service.

Asset Management System: Software that manages the production and use of enterprise technology assets, including, but not limited to, services.

Classifier: A value from a *classifier system*. In the context of WSRR, classifiers are associated with entities to give them semantic meaning.

Classifier System: A collection of classifier values that are semantically related. Examples include the NAICS industry classifier system, UNSPSC product and services classifier system, user-defined type systems, and user-defined state systems.

CMDB: Configuration Management Data Base. A metadata repository for components of an information system and their relationships and configurations.

Concept: In the context of the WSRR, this is the non-technical term for an entity in the WSRR that has no physical document. Concepts can represent groupings which can be typed, managed, and governed as a unit, and can represent external content which resides in a separate repository.

Custom Entity: A Concept representing a collection of entities in the WSRR.

Descriptive Metadata: In the context of the WSRR, the properties, classifiers, and relationships between entities in the WSRR.

Document: In the WSRR this is an XML file containing descriptive information about or relating to a service. Examples are files containing WSDL, XSD, SCDL, WS-Policy. They can also be other sorts of XML document that contain descriptive information about a service.

Endpoint: A conceptual term relating to the physical document and logical model entities that ultimately contain a service address. The WSDL and SCDL documents that contain a service's address as well as counterparts in the logical model – WSDLPortType and SCDLExport – are examples of endpoints.

Entity: In the WSRR content model, a physical document, piece of the logical model derived from shredding certain physical document types, or a Generic Object.

Generic Object: The technical name given to a Concept.

Governance: Governance is the set of activities associated with: • Establishing chains of responsibility, authority and communication to empower people (decision rights) • Establishing measurement, policy and control mechanisms to enable people to carry out their roles and responsibilities.[12]

IT Governance: IT Governance is that subset of *governance* that pertains to an organization's IT processes and the way those processes support the goals of the organization. Therefore IT governance includes the decision making rights associated with IT as well as the mechanisms and policies used to measure and control the way IT decisions are made and carried out within the organization.[13]

Logical Model: The collective set of modelled decompositions of certain physical document types. The decomposed pieces that comprise the logical model are entities which can have properties, classifiers and relationships associated with them. WSDL, XSD, SCDL, and WS-Policy to a lesser degree, all have models.

Metadata Artefact: See *Artefact*.

Ontology: A specification of concepts and the relationships that may exist between those

[12] http://en.wikipedia.org/wiki/SOA_Governance
[13] http://en.wikipedia.org/wiki/SOA_Governance

concepts for some universe of discourse or community (see also taxonomy). [14] The WSRR uses simple ontologies expressed in *OWL* as a means for describing *Classifier System*s.

OWL: Web Ontology Language. The language used to capture *Classifier System*s for use by WSRR.

Physical Artefact: A *Document* in the WSRR.

Physical Entity: A *Document* in the WSRR.

Physical Model: The set of *physical document* types stored in the WSRR. Well known types of documents that comprise the physical model include WSDL, XSD, SCDL, WS-Policy. Other XML document types can be stored in the WSRR and given a user-defined type.

Property: A name-value pair that can be associated with any entity in the WSRR. Properties can be system-defined and assigned or can be user-defined.

RAS Repository: A repository of content (assets) that conform to the Reusable Asset Specification. Assets may or may not be related to services.

Relationship: A correlation between a 'from' entity and a 'to' entity that is given a name. Relationships can be system-defined and assigned, such as through wsdl:imports, or can be user-defined.

SCDL Document: An XML document that contains all or part of a Service Component Architecture (SCA) definition of a service. SCDL documents are modelled in WSRR and are thus shredded into a logical model whose parts that can receive property, classifier, and relationship annotations, and which can be used in fine-grained queries.

Semantic Annotations: Metadata, such as classifiers, properties and relationships, that is assigned to entities to provide semantic meaning.

Service Description: The set of entities and semantic annotations that collectively define a service.

Semantic Description Metadata: See *Semantic Annotations.*

Service Endpoint: See *Endpoint*.

Service Interaction Endpoint: See *Endpoint*.

Service Metadata: All the metadata that makes up a Service Description, including metadata about a service that might not reside in WSRR but can be referenced from WSRR.

Service Metadata Artefact: See *Artefact*.

Service Metadata Artefact Type: A kind of document in the WSRR *Physical Model*. Examples of artefact type are WSDL, XSD, SCDL, WS-Policy, but can include user-defined values as well.

Service Orientation: A way of integrating your business processes as linked services, and the outcomes that these services bring.[15]

Service Oriented Architecture: An architectural style that supports service orientation.[16]

SOA: Service Oriented Architecture.

SOA Governance: A sub-discipline of *IT Governance* pertaining to the lifecycle of

[14] http://en.wikipedia.org/wiki/Ontology_%28computer_science%29
[15] ftp://ftp.software.ibm.com/software/solutions/pdfs/SOA_g224-7540-00_WP_final.pdf
[16] ftp://ftp.software.ibm.com/software/solutions/pdfs/SOA_g224-7540-00_WP_final.pdf

services, composite applications and other artifacts and processes associated with the adoption of Service-Orientation by an organization.[17]

SOAP: SOAP is a standard for exchanging XML-based messages over a computer network, normally using HTTP. SOAP forms the foundation layer of the web services stack, providing a basic messaging framework that more abstract layers can build on.[18] The WSRR offers a Web Service version of its Application Programming Interface (API) that utilizes SOAP over HTTP.

State: In the context of WSRR, the life-cycle stage of documents and Concepts. Decomposed pieces of the logical model that are derived when physical documents are shredded are not separately assigned state (but they do inherit the state of their "parent" document).

Technical Details: The what and how of a Service Description, captured by documents

Transition: The governed movement of a governed entity from one life-cycle stage to another.

UDDI: Universal Description Discovery and Integration. An early standard in the Web Services space that provides a content model and Web Service API for advertising and finding service descriptions.

User-defined Property: A *Property* that is defined by a user rather than the system.

User-defined Relationship: A *Relationship* that is defined by a user rather than the system.

Web Service: A software component that is described via WSDL and is capable of being accessed via standard network protocols such as but not limited to SOAP over HTTP.[19]

WSDL Document: An XML document that contains all or part of a description of a Web service. WSDL 1.1 documents are modelled in WSRR and are thus shredded into a logical model whose parts that can receive property, classifier, and relationship annotations, and which can be used in fine-grained queries.

WSRR: WebSphere Service Registry and Repository

WS-Policy Document: An XML document that contains all or part of a policy declaration. WS-Policy documents are only minimally modelled in WSRR, without regard for various policy dialects, and are thus only minimally shredded into a logical model whose parts that can receive property, classifier, and relationship annotations, and which can be used in fine-grained queries.

XML Schema: An *XSD Document*.

XSD Document: An XML document that contains all or part of an XML Schema. XSD documents are modelled in WSRR and are thus shredded into a logical model whose parts that can receive property, classifier, and relationship annotations, and which can be used in fine-grained queries.

[17] http://en.wikipedia.org/wiki/SOA_Governance
[18] http://en.wikipedia.org/wiki/SOAP
[19] http://www.oasis-open.org/committees/wsia/glossary/wsia-draft-glossary-03.htm