Java™ API for XML Registries (JAXR)

Proposed Final Draft: 4/10/2002

This version: JAXR Version 1.0

Please send technical comments to: jaxr-experts@east.sun.com

Please send business comments to: jaxr-business@east.sun.com

Farrukh Najmi <Farrukh.Najmi@sun.com>

SUN IS WILLING TO LICENSE THIS SPECIFICATION TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS LICENSE AGREEMENT ("AGREEMENT"). PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE CAREFULLY. BY DOWNLOADING THIS SPECIFICATION, YOU ACCEPT THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ITS TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THIS PAGE AND THE DOWNLOADING PROCESS WILL NOT CONTINUE.

Java™ API for XML Registries (JAXR) Specification ("Specification")
Version: 1.0
Status: Proposed Final Draft
Release: 4/10/2002

901 San Antonio Road, Palo Alto, California 94303, U.S.A.

NOTICE

1. The Specification is protected by copyright and the information described therein may be protected by one or more U.S. patents, foreign patents, or pending applications. Except as provided under the following license, no part of the Specification may be reproduced in any form by any means without the prior written authorization of Sun Microsystems, Inc. ("Sun") and its licensors, if any. Any use of the Specification and the information described therein will be governed by the terms and conditions of this Agreement. By viewing, downloading or otherwise copying the Specification, you agree that you have read, understood, and will comply with all of the terms and conditions set forth herein.
2. Sun hereby grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense), under its applicable intellectual property rights to view, download, use and reproduce the Specification ("Specification") only for the purpose of evaluation, which shall be understood to include developing applications intended to run on an implementation of the Specification provided that such applications do not themselves implement any portion(s) of the Specification.

3. Sun also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights or patent rights it may have therein to create and/or distribute an Independent Implementation. Such license will authorize the creation and distribution of Independent Implementations provided such Independent Implementations: (i) fully implement the Spec(s) including all its required interfaces and functionality; (ii) do not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Spec or Specs being implemented; (iii) pass the TCK for such Spec; and (d) are capable of operating on a Java platform which is certified to pass the complete TCK for such Java platform, unless the Spec itself is for a Java platform, in which case this subparagraph (iv) shall not apply. This subparagraph (iv) also shall not apply to those portions of the Specification (if any) that define network protocols that are not specific to a Java platform. No license is granted hereunder for any other purpose. This license will terminate immediately without notice from Sun if you fail to comply with (and do not promptly cure) any material provision of or act outside the scope of this license.

4. You need not include requirements (a)-(d) from the previous paragraph or any other particular "pass through" requirements in any license you grant concerning the use of your Independent Implementation or products derived from it. However, except with respect to implementations of the Specification (and products derived from them) by the your licensee that satisfy requirements (a) through (d) from the previous paragraph, you may neither: (i) grant or otherwise pass through to your licensees any licenses under Sun's applicable intellectual property rights; nor (ii) authorize your licensees to make any claims concerning their implementation's compliance with the Specification.

5. For the purposes of this Agreement, Independent Implementation shall mean an implementation of the Specification that neither derives from any of Sun's source code or binary code materials nor, except with an appropriate and separate license from Sun, includes any of Sun's source code or binary code materials.

TRADEMARKS
No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun's licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DISCLAIMER OF WARRANTIES THE SPECIFICATION IS PROVIDED "AS IS". SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT

THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. SUN MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java application, applet and/or clean room implementation; and/or (iii) any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND
U.S. Government: If this Specification is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

EXPORT CONTROL
You acknowledge and agree that the Specification is subject to the U.S. Export Administration Laws and Regulations. Diversion of such information contrary to U.S. law is prohibited. You agree that none of the information contained in the Specifications, nor any direct product therefrom, is being or will be acquired for, shipped, transferred, or re-exported, directly or indirectly, to proscribed or embargoed countries or their nationals, nor be used for nuclear activities, chemical biological weapons, or missile projects unless authorized by the U.S. Government. Proscribed countries are set forth in the U.S. Export Administration Regulations. Countries subject to U.S. embargo are: Cuba, Iran, Iraq, Libya, North Korea, Syria, Serbia, and the Sudan. This list is subject to change without further notice from Sun, and you must comply with the list as it exists in fact. You certify that you are not on the U.S. Department of Commerce's Denied Persons List or affiliated lists or on the U.S. Department of Treasury's Specially Designated Nationals List. You agree to comply strictly with all U.S. export laws and assume sole responsibility for obtaining licenses to export or re-export as may be required

REPORT
You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your use of the Specification ("Feedback"). To the extent that you provide Sun with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

GENERAL INFORMATION
This Agreement constitutes the entire agreement between you and Sun and governs your use of the Specification, superseding any prior agreements between you and Sun. You also may be subject to additional terms and conditions that may apply when You use other Sun services, third-party content or third-party software. You must not assign or otherwise transfer the Agreement nor any right granted hereunder.
California law and controlling U.S. federal law govern any action related to the Agreement . No choice of law rules of any jurisdiction apply. The parties specifically disclaim the U.N. Convention on Contracts for the International Sale of Goods. You and Sun agree to submit to the personal and exclusive jurisdiction of the courts located within the county of Santa Clara, California.
The failure of Sun to exercise or enforce any right or provision of the Agreement does not constitute a waiver of such right or provision. If a court of competent jurisdiction finds any provision of the Agreement to be invalid, the parties nevertheless

agree that the court should endeavor to give effect to the parties' intentions as reflected in the provision, and the other provisions of the Agreement remain in full force and effect. The section titles in the Agreement are for convenience only and have no legal or contractual effect.

Table of Contents

Table of Figures

417

Introduction

1.1 Status of this Document

This specification is being developed following the Java™ Community Process℠ (JCP ℠ 2.0). Comments from Experts, Participants, and the broader Java Developer community have been reviewed and incorporated into this specification.

This document is the JAXR Specification, version 1.0 and is the final work item of the JSR093 Expert Group (EG).

This document has been designated as Final Release.

1.2 Abstract

This document defines the objectives and functionality for Java API for XML Registries or JAXR.

Currently there are numerous overlapping specifications for business registries. Examples include ISO 11179, OASIS, eCo Framework, ebXML and UDDI. JAXR provides a uniform and standard API for accessing such registries within the Java platform.

1.3 General Conventions

1. The term "*registry provider"* is used to describe implementations of business registries conforming to various registry specifications and emerging standards.
2. The term "*JAXR provider"* is used to describe implementations of the JAXR API. A JAXR provider provides access to a specific registry provider or to a class of registry providers that are based on a common specification.
3. The term "*JAXR client"* is used to describe client programs that access business registries using the JAXR API.
4. The term *"repository item"* is used to refer to actual content (e.g. an XML Schema document, as opposed to metadata about the XML Schema document) submitted to a registry. The term *"repository item instance"* is used to refer to a single instance of some repository item.
5. The term *"registry object"* is used to refer to metadata that catalogs or describes a repository item. It is reflected by the RegistryObject interface in the JAXR information model and its sub-interfaces.

6. The verb *"catalogs"* is often used when describing metadata classes. For example, the statement "Class A catalogs B" is equivalent to the statement "Class A provides metadata for B".
7. This document does not include the complete API documentation generated by the Javadoc™ software. Partial API documentation fragments are included occasionally to facilitate understanding. The reader is expected to read the complete API documentation as a companion to this document.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

1.4 Target Audience

The target audience for this specification is the community of software developers who are:

1. Implementers of JAXR providers
2. Implementers of JAXR clients

1.5 JAXR Expert Group

The JAXR specification is the result of a collaborative effort and collective wisdom of the JSR093 Expert Group and the companies and individuals who have supported this work with their participation in the Java Community.

Joseph Baran - Extol, Inc.
Ben Bernhard - IONA
Marco Carrer - Oracle
Alex Ceponkus - Bowstreet
Joel Farrell - IBM Corporation
Tom Gaskins - Hewlett-Packard Company
Wooyoung Kim - Individual
Amelia A. Lewis - Tibco Extensibility Inc.
Sam Lee - Oracle
Dale Moberg - Cyclone Commerce
Farrukh Najmi - Sun Microsystems
Eric Newcomer - IONA Technologies
Sanjay Patil - IONA Technologies
Will Raymond - Tibco Extensibility Inc.
Waqar Sadiq - EDS
Krishna Sankar - CISCO
Nikola Stojanovic - Encoda Systems, Inc.
Omar Tazi - webGain

Ravi Trivedi - Hewlett-Packard Company
Lyndon Washington - SilverStream
Prasad Yendluri - webMethods Corporation
Phil Zimmerman - BEA Systems

1.6 Acknowledgements

Graham Hamilton, Mark Hapner, Eduardo Pelegri-Llopart, Bill Shannon, Robert Bissett, Irene Caruso, Peter Eakle, Joe Fialli, Maydene Fisher, Kim Haase, Steve Hanna, Peter Kacandes, Nandkumar Kesavan, Tom Kincaid, Ramesh Mandava, Bhakti Mehta, Ron Monzillo, Kevin Osborn, Cecilia Peltier, Karen Schaffer, Leslie Schwenk, Karen Shipe, Christine Tomlinson, Sekhar Vajjhala, Peter Walker, Kathy Walsh, (all from Sun Microsystems) have all made invaluable contributions to the JAXR 1.0 specification. Thanks to Jeff Jackson, Karen Tegan and Connie Weiss for their sponsorship and support of JAXR.

1.7 Relationship to Other Java APIs

JAXR is related to several other Java APIs for XML. In future, these APIs may become part of the Java 2 Platform, Enterprise Edition (J2EE™ platform).

1.7.1 JAXP

Java API for XML Processing or JAXP enables flexible XML processing from within Java programs.

The JAXR API will make direct XML processing less important for JAXR clients. However, JAXP may be used by implementers of JAXR providers and JAXR clients for processing XML content that is submitted to or retrieved from the registry. The JAXP API is likely to also be used in implementations of the JAXB API described next.

1.7.2 JAXB

Java API for XML Binding or JAXB enables simplified XML processing using Java classes that are generated from XML schemas.

The JAXR API will make direct XML processing less important for JAXR clients. However, JAXB may be used by implementers of JAXR providers and JAXR clients for processing XML content that is submitted to or retrieved from the registry.

1.7.3 JAX-RPC

Java API for XML-based RPC or JAX-RPC provides an API for XML-based RPC communication in the Java platform.

Implementations of the JAXR providers may use JAX-RPC for communication between JAXR providers and registry providers that export a SOAP-based RPC like interface (e.g. UDDI).

1.7.4 JAXM

Java API for XML Messaging or JAXM provides an API for packaging and transporting of message based business transactions using on-the-wire protocols defined by emerging standards.

Implementations of the JAXR providers may use JAXM for communication between JAXR providers and registry providers that export an XML Messaging based interface (e.g. ebXML TRP).

1.8 Design Objectives

This section describes the high level design objectives for the JAXR API.

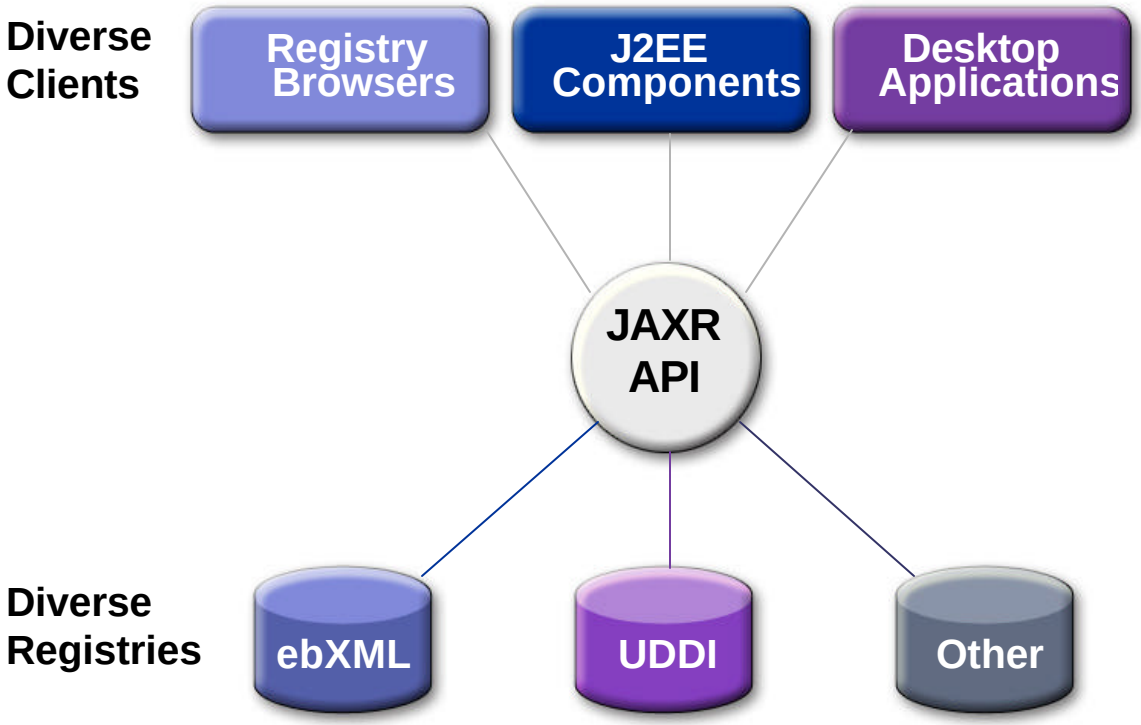
1.8.1 Goals

The goals of this version of the specification are to:

1. Define a general purpose Java API for accessing business registries that allows any JAXR client to access and interoperate with any business registry that is accessible via a JAXR provider.
2. Define a pluggable provider architecture that enables support for diverse registry specifications and standards.
3. Support a union of the best features of dominant registry specifications rather than a common intersection of features. *JAXR is not a least common denominator API*.
4. Ensure support for dominant registry specifications such as ebXML and UDDI, while maintaining sufficient generality to support other types of registries, current or future.
5. Ensure synergy with other Java specifications related to XML.

Figure 1 below shows how diverse JAXR clients can interoperate with diverse registries using the JAXR API.

552



553

554 **Figure 1: Interoperability between diverse JAXR clients and diverse registries**

555 ### 1.8.2 Non Goals

556 This specification does not aim to define either business registry standards or
557 XML messaging standards. These standards belong in standards bodies such as
558 OASIS, W3C or IETF. Instead, this specification aims to define standard Java
559 APIs to allow convenient access from Java to emerging registry standards.

560 ## 1.9 Caveats and Assumptions

561 It is assumed that:

562 1. The reader is familiar with UML notation. UML notation is used throughout
563 this document for most of the diagrams.

564

2 Overview

2.1 What Is a Registry

Most business-to-business (B2B) interactions are based on a collaborative process between 2 parties that are engaged in a partnership. A registry is a neutral 3rd party that helps facilitate such collaboration. A registry is available to organizations as a shared resource often in the form of a web based service. A registry is a key component in any Web Services architecture because it provides organizations with the ability to publish, discover and utilize web services. Registries enable dynamic and loosely coupled B2B collaboration.

[Note] While this document may present registry use cases in a business-focused context, the JAXR API is sufficiently general to support many other types of use cases.

2.2 Registry Use Case Scenarios

Figure 2 below illustrates a few of the common use case scenarios involving a business registry. The scenario shows how a registry facilitates a buyer company discovering a seller company and engaging in a collaborative B2B process.

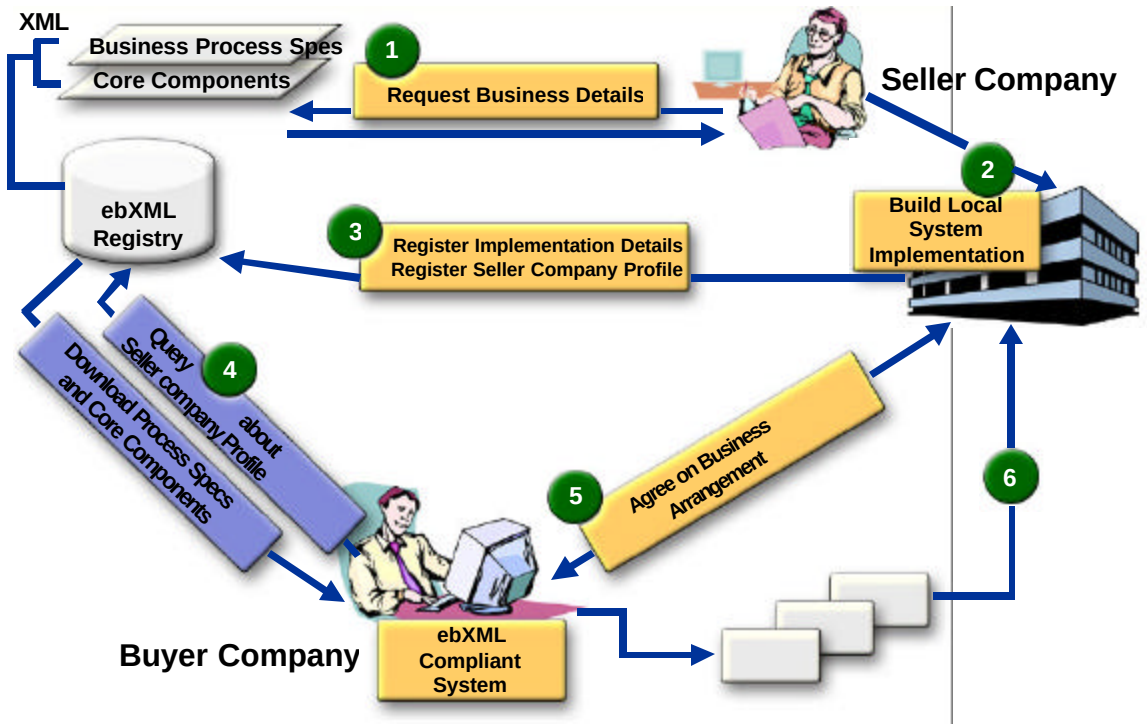


Figure 2: A Registry Use Case Scenario

1. First, the Seller company queries a registry for specifications defining a collaborative business process as well as core components that define reusable XML elements used in business documents (e.g. Address, Contact etc.). These specifications have previously been submitted by a vertical standards organization.
2. The Seller then uses the specifications and core components downloaded from the registry to implement their local eBusiness system with support for the desired collaborative processes.
3. The Seller then registers information about their company, their products and their services in the registry. Such information may be classified to facilitate discovery by potential buyers.
4. A Buyer company may browse the registry by classifications etc. and discover the Seller. They may also download technical specifications and core components to implement their local system to support the collaborative process.
5. The Buyer then negotiates with the Seller on an agreement to collaborate in the chosen collaborative process implemented and agreed to by both sides.

602 6. The two parties finally engage in the desired collaborative business
603 process and exchange business documents.

604 2.3 Participant Roles

605 This section describes the key roles played by participants (actors) within various
606 registry use case scenarios. This section is not a complete list of roles.

607 2.3.1 Submitting Organization

608 A submitting organization (SO) is an organization that submits or publishes
609 content to a registry. An SO may be an enterprise or an entity within an
610 enterprise.

611 An SO owns the content that it publishes to a registry.

612 2.3.2 Content Submitter

613 A content submitter is a user who belongs to a submitting organization and is
614 authorized to submit content on behalf of the organization.

615 2.3.3 Registry Operator

616 A registry operator is responsible for operating a registry. A registry operator has
617 special access control and authorization privileges within the registry under their
618 operation.

619 2.3.4 Registry Guest

620 A registry guest is a non-privileged casual user of the registry who simply
621 browses the data within the registry.

622 2.4 Registry Vs. Repository

623 The terms registry and repository are often used together and sometimes
624 confused with each other.

625 The following sections describe the distinction between a registry and a
626 repository and introduce the content of each.

2.4.1 Repository and Repository Items

Information published by an SO to a registry is stored in a stable store called a repository. The registry maintains the repository. The repository is the holder of content (e.g. DTDs, XML Schemas, WSDL documents etc.) submitted by an SO to a registry. Instances of the content stored in the repository are called *repository items*.

The JAXR API does not directly provide access to the repository. Instead, all access to the repository is through the registry. As such, the repository is an implementation detail of a registry. It is mentioned in this specification only as a concept. However, neither the repository nor repository items are part of the JAXR information model or API.

2.4.2 Registry and Registry Objects

When an SO submits repository items using the JAXR API, it also provides additional metadata that *catalogs* or *describes* the repository items. Such metadata is referred to as *registry objects* in the JAXR information model.

In summary, a repository is a holder of submitted content while a registry is a catalog that describes the submitted content in the repository. It should be noted that not all registries include the repository functionality.

2.5 Functionality of a Registry

This section describes the functionality that is provided by a registry.

2.5.1 Registry as Electronic Yellow Pages

Registries facilitate the creation of business relationships by providing an independent online information exchange service that allows service providers (e.g. sellers) to advertise their products and services, and service consumers (e.g. buyers) to discover these products and services. Such an information exchange service is sometimes referred to as “electronic yellow pages”.

2.5.1.1 Flexible Classification Capability

Registries provide a rich classification capability that allows content providers to classify content such as organization and service descriptions in arbitrary and flexible ways. For example, content submitted to the registry may be the description of a business organization that is classified by the *industry* it belongs to, the *geography* it is located in, the *business processes* it supports, and the *products* it sells.

660 Such flexible classification capabilities of registries facilitate discovery of content
661 by interested parties.

662 ### 2.5.2 Registry as a Database of Relatively Static Data

663 A registry (and its repository) stores metadata and data. As such, it is much like a
664 database. It stores information about:

665 o Collaborative business process descriptions that describe in XML form a
666 specific business protocol (e.g. RosettaNet PIP3A4 for purchase orders)

667 o Parties in a collaborative business process

668 o XML Schemas that define the structure of the XML documents exchanged
669 during a collaborative Business Process

670 A registry plays a role in B2B applications that is similar to that played by
671 databases in enterprise applications; it provides a way for applications to store
672 relatively static information reliably and to enable sharing of such information.

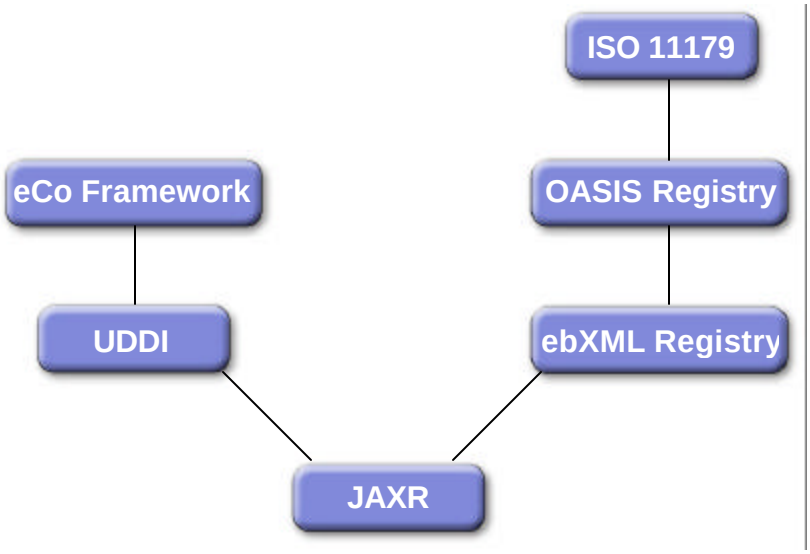
673 ### 2.5.3 Registry as Electronic Bulletin Board

674 A registry may also provide means to exchange dynamic content between
675 parties. Examples include generic event notification, price changes, discounts,
676 promotions etc. Such dynamic capabilities allow for more Just-In-Time B2B
677 partnerships.

678 ## 2.6 Existing Registry Specifications

679 Currently there are numerous registry specifications. Examples include OASIS,
680 eCo Framework, ebXML and UDDI. While there may be some similarity between
681 these specifications, in general these are diverse specifications.

682 JAXR API aims to be the confluence of the various registry specifications as
683 shown in Figure 3 below:



684

685 **Figure 3: Registry specification family tree**

686 2.7 Registry Provider

687 A registry provider provides an implementation of a registry specification or
688 standard. Examples include:

689 - A UDDI registry provider that implements the UDDI registry specifications
690 - An ebXML registry provider that implements the ebXML Registry
691 specifications

692 A registry provider is not required to implement the JAXR specification.

693 2.8 JAXR Provider

694 A JAXR Provider provides an implementation of the JAXR specification.
695 Typically, a JAXR provider is implemented to access an existing registry
696 provider.

697 2.9 JAXR Client

698 A JAXR client is a Java program that uses the JAXR API to access the services
699 provided by a JAXR provider.

700 2.10 Support for Multiple Registry Specifications

701 The JAXR API must support diverse registry specifications that vary significantly
702 in their capabilities and underlying information model.

JAXR aims to support diverse registry specifications by providing a broad and capable API rather than a “least-common-denominator” API. To use a Java analogy, the JAXR philosophy is akin to designing a Java Virtual Machine that is a union of the most useful capabilities of all supported hardware platforms rather than an intersection of their capabilities. This approach to supporting diverse registry specifications means that not all Registries are able to support all aspects of the JAXR API.

2.10.1 Capability Profiles

The JAXR API categorizes its API methods by a small number of capability profiles. Currently only two capability profiles are defined (level 0 and level1).

2.10.1.1 Assignment of Capability Level to Methods

Each method in the JAXR API is assigned a capability level. The capability level is defined in the API documentation for each method in a class or interface in the JAXR API.

2.10.1.2 Assignment of Capability Level to Interfaces and Classes

There is no assignment of capability level to interfaces and classes in the JAXR API. Capability assignment is done at the method level only.

2.10.1.3 Declaration of Capability Level by a JAXR Provider

A JAXR provider must declare the capability level for its implementation of the JAXR API. A JAXR client may discover a JAXR provider’s capability level by invoking methods on an interface named CapabilityProfile as defined by the JAXR API. If a JAXR provider declares support for a specific capability level then it implicitly declares support for lower capability levels. For example, a JAXR provider that declares support for the level 1 profile implicitly declares support for level 0 profile.

A JAXR provider must implement the functionality described by the JAXR API for each method that is assigned a capability level that is less than or equal to the capability level declared by the JAXR provider.

A JAXR provider must implement all methods that are assigned a capability level that is greater than the capability level declared by the JAXR provider, to throw an UnsupportedCapabilityException. A JAXR provider must never implement any other behavior for methods assigned a greater than the capability level declared by the JAXR provider. The reason for this restriction is that it is necessary to ensure portable behavior for JAXR clients for any JAXR provider within a specific capability level.

738 ### 2.10.2 Level 0 Profile

739 Support for the level 0 profile is required to be supported by all JAXR providers.
740 The methods assigned to this profile provide the most basic registry capabilities.

741 ### 2.10.3 Level 1 Profile

742 Support for the level 1 profile is optional for JAXR providers. The methods
743 assigned to this profile provide more advanced registry capabilities that are
744 needed by more demanding JAXR clients. Support for the Level 1 profile also
745 implies full support for the Level 0 profile.

746 ### 2.10.4 Capability Level and JAXR Clients

747 A JAXR client may be written to use only those methods that are assigned a level
748 0. Such a client is able to access any JAXR provider in a portable manner.

749 An advanced JAXR client may also be written to methods that are assigned a
750 level 1. This level 1 client is able to access only level 1 compliant JAXR providers
751 in a portable manner.

752 ## 2.11 Capability Levels and Registry Standards

753 JAXR providers for UDDI must be level 0 compliant. JAXR providers for ebXML
754 must be level 1 compliant.

755

3 Architecture

This chapter describes the high-level architecture for JAXR and introduces some of the core interfaces.

Figure 4 below shows the high-level view of the JAXR architecture. It is frequently referred to, within this specification.

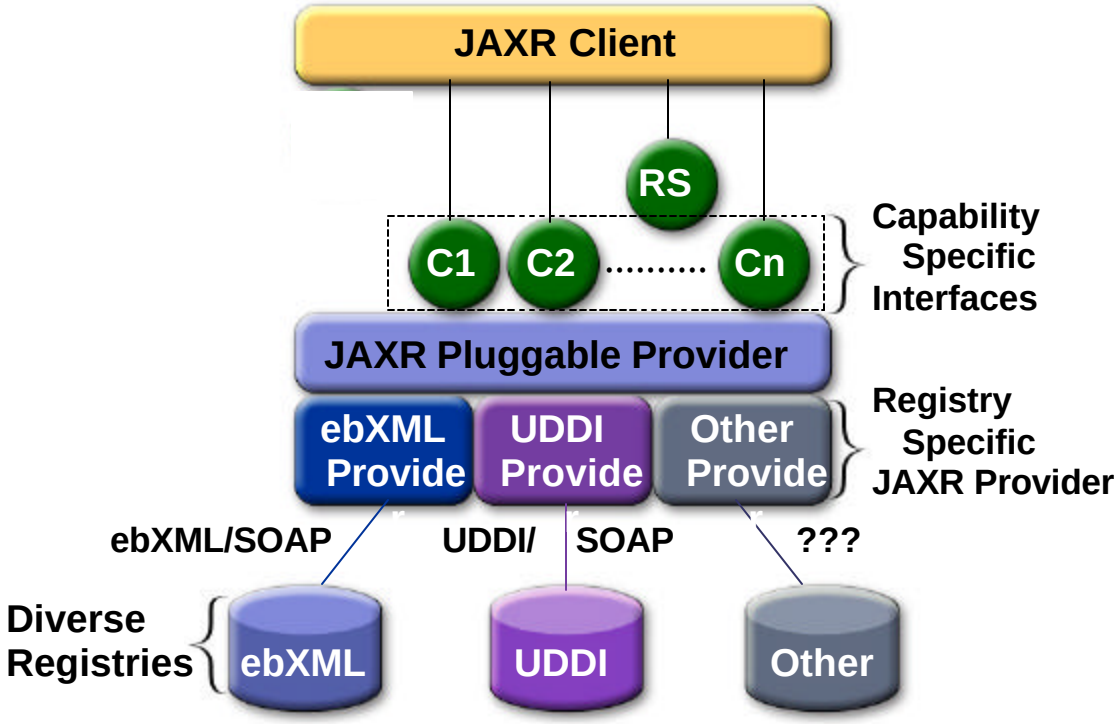


Figure 4: JAXR Architecture

The circles represent the various interfaces implemented by the JAXR client and the JAXR provider:

- RS represents the `RegistryService` interface implemented by the JAXR provider.
- C1, C2 through Cn represent the JAXR interfaces implemented by the JAXR provider that provide the various registry capabilities. These interfaces are introduced later in this specification.

[Note] The JAXR client and the JAXR provider are expected to be co-located within the same JVM process in most implementations. The only

```
            distribution point over the web is between the
            JAXR provider and the registry provider
            (between the middle and bottom tiers).
```

In the following sections, each component of the JAXR architecture is described in a top-down order. Most descriptions are accompanied by a thumbnail version of Figure 4. The thumbnail figure highlights the component being described by pointing to it with a block arrow and showing it in red color.

3.1 JAXR Client

At the top of Figure 4 is the JAXR client. The JAXR client may be any standalone Java application or an enterprise component based on J2EE technology. The JAXR client uses the JAXR API to access a registry via a JAXR provider.

3.2 Interface *Connection*

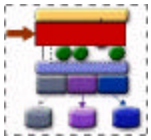


Figure 5: Interface *Connection*

A `Connection` object (pointed to by block arrow in Figure 5) represents a client session with a registry provider using a JAXR provider. It maintains state information for a specific connection.

A client must create a JAXR Connection to a registry provider using an appropriate JAXR provider in order to employ the services of that registry using the JAXR API. Chapter 7 describes the role of Connections in further detail.

The client uses the JAXR ConnectionFactory interface to create a Connection.

The JAXR Connection is not explicitly shown in Figure 4. However, the RegistryService interface defined by the JAXR API is contained within a JAXR Connection.

1. The Connection interface provides various setter methods (e.g. setSynchronous, setCredentials) that allow the JAXR client to dynamically alter its state, context, and preferences with the JAXR provider at any time.

3.3 Interface *RegistryService*

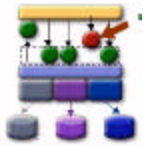


Figure 6: Interface *RegistryService*

The RegistryService interface (pointed to by block arrow in Figure 6) is the principal interface implemented by a JAXR provider. A registry client can get this interface from its Connection to a JAXR provider.

The `RegistryService` interface provides the getter methods that are used by the client to discover various capability-specific interfaces implemented by the JAXR provider. It also provides a `getCapabilityProfile` method that allows the JAXR client to access the capability profile that describes the capabilities supported by the JAXR provider.

3.4 Capability-specific Interfaces

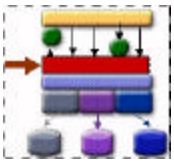


Figure 7: Capability-specific Interfaces

The capability-specific interfaces (pointed by block arrow in Figure 7) provide specific capabilities such as:

- Life cycle management, which is discussed in Chapter 8.
- Query management, which is discussed in Chapter 9.

Capability specific interfaces are usually named xxManager where xx represents the specific capability provided by that interface.

3.5 The JAXR Provider

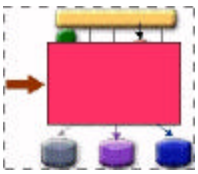


Figure 8: JAXR Provider

The JAXR provider (pointed to by block arrow in Figure 8) is an implementation of the JAXR API. A JAXR client accesses a registry via a JAXR provider.

Figure 4 shows the JAXR provider as the union of the JAXR pluggable provider and the registry-specific JAXR providers.

[Note] The following sections describing JAXR Pluggable provider, registry specific providers and JAXR bridge providers are non-normative. They describe an architectural vision for the JAXR API. However, it should be noted that the JAXR API does not define a service provider interface (SPI) for plugging registry specific providers into a JAXR pluggable provider. This SPI is deferred to a future release of the JAXR specification.

3.5.1 JAXR Pluggable Provider

The JAXR Pluggable provider implements features of the JAXR API that are independent of any specific registry type. The Pluggable provider provides a single abstraction for multiple registry-specific JAXR providers. It allows the client to avoid being exposed to the fact that there are multiple registry-specific JAXR providers performing the actual registry access.

An important feature of the JAXR Pluggable provider is providing a Pluggable ConnectionFactory implementation that can create JAXR Connections using the appropriate registry-specific JAXR provider.

3.5.2 Registry-specific JAXR Provider

The registry-specific JAXR providers implement the JAXR API in a registry-specific manner. A registry-specific JAXR provider plugs into the JAXR Pluggable provider and is used by the JAXR Pluggable provider in a delegation pattern. The contract between the JAXR Pluggable provider and a registry-specific JAXR provider is currently not defined. It will be defined in a future version of this specification.

A registry-specific JAXR provider accepts JAXR requests from the client and transforms them into equivalent requests based on the specifications of the target registry. It dispatches the registry-specific requests to the registry provider using registry-specific protocols.

A registry provider processes a request from the registry-specific JAXR provider and sends back a registry-specific response. The registry-specific JAXR provider then transforms the registry-specific response into an equivalent JAXR response that is delivered to the JAXR client.

From the registry provider's perspective, its client is the registry-specific JAXR provider.

3.5.3 JAXR Bridge Providers

A JAXR Bridge Provider is a type of registry-specific JAXR Provider.

It is likely that most registry-specific JAXR providers will be developed as bridges to existing registry providers. Such JAXR providers are referred to as JAXR Bridge providers. A bridge provider is not specific to any particular registry instance. Instead, a bridge provider is specific to a class of registries (e.g. OASIS ebXML registry, UDDI registry) and may be used to access any registry instance that is compliant with the specification that defines that class of registries.

For example, an ebXML Bridge provider gives access to any ebXML compliant registry implementation, while a UDDI bridge provider gives access to any UDDI compliant registry implementation.

3.6 Registry Provider

Registry providers are shown as the bottom layer in Figure 4. These are implementations of various registry specifications such as ebXML and UDDI.

3.7 JAXR API Package Structure

The JAXR API is divided into two main packages:

1. The `javax.xml.registry.infomodel` package consists of interfaces that define the information model for JAXR. These interfaces define the types of objects that reside in a registry and how they relate to each other. The information model is discussed in detail in Chapter 4.
2. The `javax.xml.registry` package consists of the interfaces and classes that define the registry access interface.

While the information model describes what types of objects reside in the registry, the access interfaces in `javax.xml.registry` package define how those objects are submitted to the registry and subsequently managed. Figure 10 shows the interfaces and classes defined by the JAXR API as defined by the `java.xml.registry` package. The information model interfaces are described in detail in Chapter 4.

3.7.1 Responses and Exceptions

Figure 9 shows the various interfaces and classes defined by the JAXR API for communicating Exceptions and responses. The RegistryException class is the common base class for all JAXRExceptions that represent exceptions and errors that occurred on the registry provider side rather than the JAXR provider side during a JAXR API call.

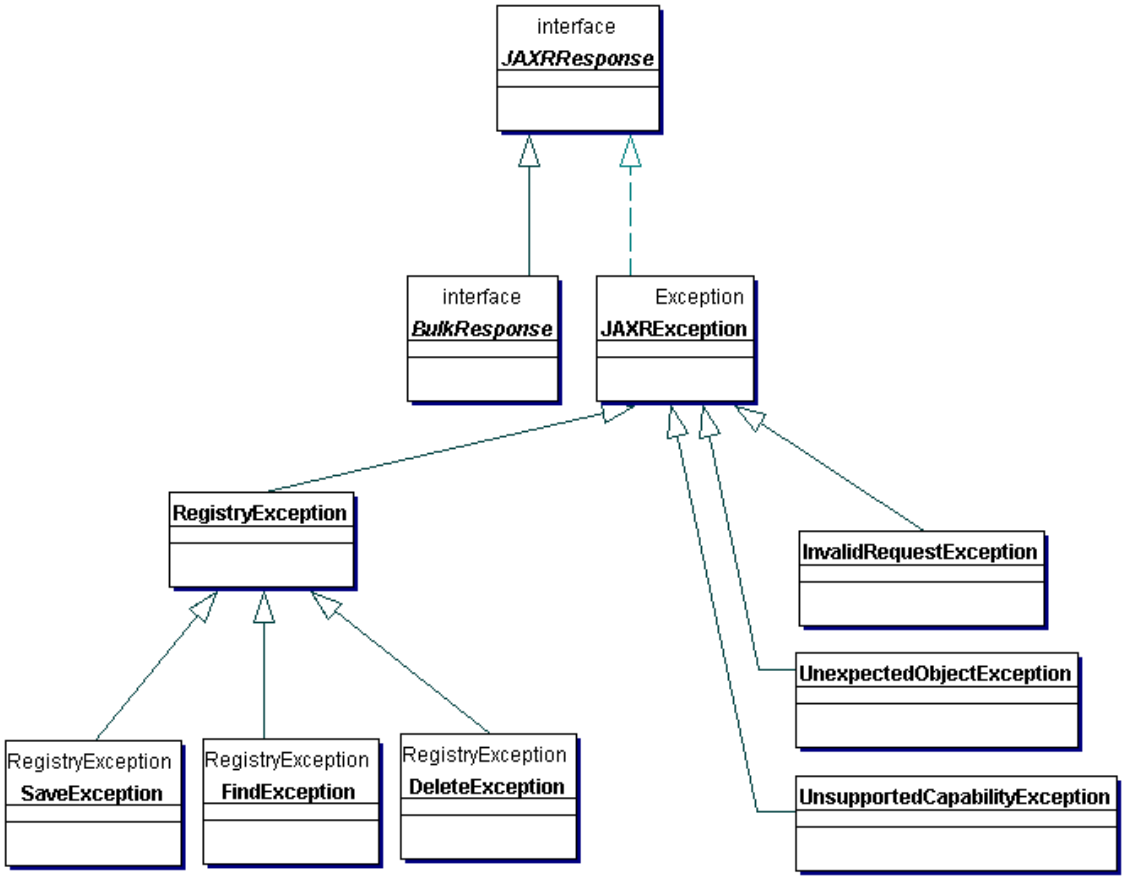
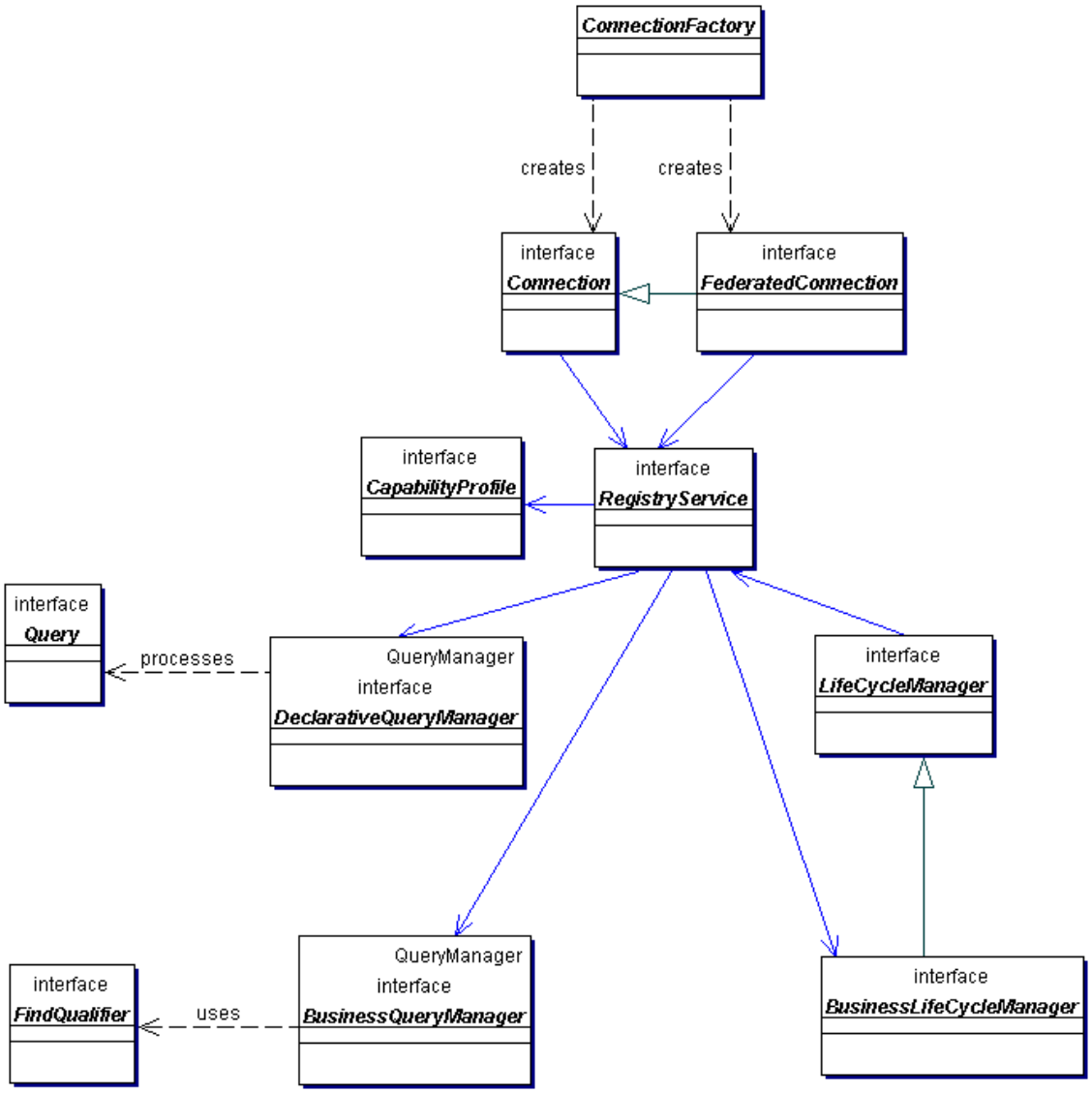


Figure 9: JAXR Responses and Exceptions

3.7.2 Main Interfaces

Figure 10 shows the main interfaces defined by the JAXR API. Part of the API is identified as the *Query API* while part of it is identified as *Life Cycle Management API*. Also part of the API is identified as providing a Business focused API while part of it is identified as providing a more generic API. Details of these interfaces will be provided later in this specification or in the API documentation for the JAXR API.



906

907 **Figure 10: Main Interfaces defined by the JAXR API**

908

909

4 Information Model

The JAXR information model is largely based on the ebXML Registry Information model as defined by [RIM] and extended to add concepts borrowed from UDDI as defined by [UDDI-DS]. A normative binding to both [RIM] and [UDDI-DS] is defined in Appendix C and Appendix D.

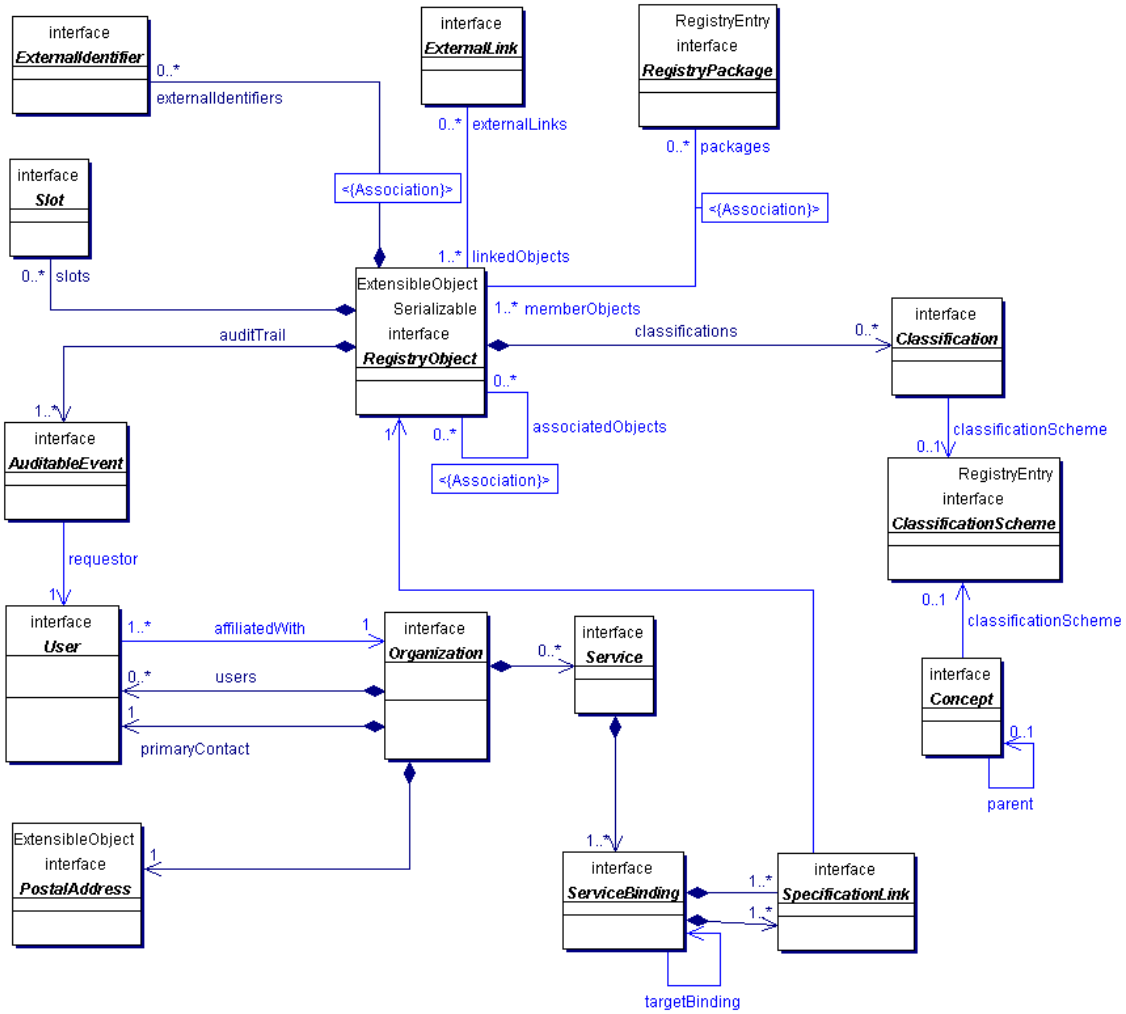
The information model related interfaces are defined in the JAXR package `java.xml.registry.infomodel`. These interfaces may be viewed as providing a simple Java binding to a unified information model from the dominant registry specifications. The JAXR information model is the confluence of these registry specifications.

4.1 Information Model: Public View

This section provides a high-level public view of the most visible objects in the registry.

Figure 11 shows the public view of the objects in the registry and their relationships as a UML class diagram. It does not show inheritance, class attributes or class methods.

The reader is reminded that the information model does not model actual repository items.



928

929 **Figure 11: Information Model Public View**

930 The following sections provide high-level information on the information model
931 interfaces. More background and detail may be found in the API documentation.

932 ### 4.1.1 RegistryObject

933 The RegistryObject class is an abstract base class used by most classes in the
934 model. It provides minimal metadata for registry objects. It also provides methods
935 for accessing related objects that provide additional dynamic metadata for the
936 registry object.

4.1.2 Organization

Organization instances are RegistryObjects that provide information on organizations such as a Submitting Organization. Each Organization instance may have a reference to a parent Organization. An Organization may have a set of Service instances.

4.1.3 Service

Service instances are RegistryObjects that provide information on services (e.g. web services) offered by an Organization. A Service may have a set of ServiceBinding instances.

4.1.4 ServiceBinding

ServiceBinding instances are RegistryObjects that represent technical information on a specific way to access a specific interface offered by a Service instance. A ServiceBinding may have a set of SpecificationLink instances.

4.1.5 SpecificationLink

A SpecificationLink provides the linkage between a ServiceBinding and one of its technical specifications that describes how to use the service using the ServiceBinding. For example, a ServiceBinding may have a SpecificationLink instance that describes how to access the service using a technical specification in the form of a WSDL document or a CORBA IDL document.

4.1.6 ClassificationScheme

A ClassificationScheme instance represents a taxonomy that may be used to classify or categorize RegistryObject instances.

A very common example of a classification scheme in science is the Classification of living things where living things are categorized in a tree-like structure. Another example is the Dewey Decimal system used in libraries to categorize books and other publications. ClassificationScheme is described in detail in Chapter 5. A common example in eBusiness is the North American Industry Classification System (NAICS), which is a classification scheme used to classify businesses and services by the industry to which they belong.

4.1.7 Classification

Classification instances are used to classify a RegistryObject instance using a classification scheme. The ability to classify RegistryObjects in a registry is one of the most significant features provided by a registry. This is because classification facilitates rapid discovery of RegistryObjects within the registry. Classification is described in detail in chapter 5.

4.1.8 Concept

A Concept instance represents an arbitrary notion (or concept). It can be virtually anything. While concepts may be used for many purposes, the following list summarizes some of the main uses of Concepts at this time:

1. Concepts may be used to define the hierarchical tree structure and detailed elements of a classification scheme as described earlier in Section 4.1.6. The root of the tree structure is defined by the ClassificationScheme instance while descendent nodes in the tree structure are Concept instances. This use is described in detail in section 5.
2. Concepts may be used to define extensible enumerations for use in values for certain attributes (e.g. objectType attribute in RegistryObject). This is essentially a special case of defining the structure of a classification scheme. This use is described in Appendix A.
3. Concepts may be used to serve as a proxy for content that is external to a level 0 registry by providing a unique ID for the external content. This is similar to the role played by tModels in UDDI when used for the purposes of providing a technical finger print for content external to the UDDI registry such as a WSDL document.
4. Concepts may be used to define namespaces for external identifiers such as DUNS.

4.1.9 Association

Association instances are used to define many-to-many associations between objects in the information model. Associations are described in detail in chapter 6.

4.1.10 RegistryPackage

RegistryPackage instances are used to group logically related RegistryObjects together. A RegistryPackage may contain any number of RegistryObjects. A RegistryObject may be a member of any number of RegistryPackages.

4.1.11 ExternalIdentifier

ExternalIdentifier instances provide identification information to a RegistryObject. Such identification may be based on well-known identification schemes such as DUNS number and Social Security Number. Such identification may also be based on proprietary identification schemes. The JAXR information model reuses the ClassificationScheme class for representation of identification schemes.

4.1.12 ExternalLink

ExternalLink instances provide a link to content that is managed outside the registry using a URI to the external content. This URI is contained within the externalURI attribute of ExternalLink. Unlike content managed in the repository, such external content may change or be deleted at any time without the knowledge of the registry. A RegistryObject may be associated with any number of ExternalLinks.

Consider the case where a Submitting Organization submits a repository item (e.g. a WSDL document) and wants to associate some external content with that object (e.g. the Submitting Organization's home page). The ExternalLink enables this capability. A potential use of the ExternalLink capability may be in a GUI tool that displays the ExternalLinks defined for a RegistryObject. The user may click on such links and navigate to an external web page referenced by the link.

When a JAXR client sets the externalURI attribute in an ExternalLink, either by a LifeCycleManager.createExternalLink call, or the ExternalLink.setExternalURI call, the JAXR provider must check if it is an HTTP URL. If so, the provider must validate that the HTTP URL points to a valid and accessible resource. If the HTTP URL is found to be invalid or inaccessible, the JAXR provider must throw an InvalidRequestException.

4.1.13 Slot

Slot instances provide a dynamic way to add arbitrary attributes to RegistryObject instances at runtime. This ability to add attributes dynamically to RegistryObject instances enables extensibility within the information model.

4.1.14 ExtensibleObject

The interface ExtensibleObject is extended by most interfaces in the JAXR information model. It provides methods that allow the addition, deletion and lookup of Slot instances. The ExtensibleObject interface provides extensibility to the JAXR information model.

4.1.15 AuditableEvent

AuditableEvent instances are RegistryObjects that are used to provide an audit trail for RegistryObjects. AuditableEvent is described in detail in section 4.2.1.

4.1.16 User

User instances are RegistryObjects that are used to provide information about registered users within the registry. Each User is affiliated with an Organization. User objects are used in the audit trail for a RegistryObject.

4.1.17 PostalAddress

PostalAddress defines attributes of a postal address. Currently, it is used to provide address information for a User and an Organization.

4.2 Information Model: Inheritance View

Figure 12 shows the inheritance or “is a” relationships between the classes in the information model. Note that it does not show the other types of relationships, such as “has a” relationships, since they have already been shown in Figure 11. Class attributes and class methods are also not shown. Detailed descriptions of methods and attributes of most interfaces and classes are available in the JAXR API documentation.

The reader is again reminded that the information model does not model actual repository items.

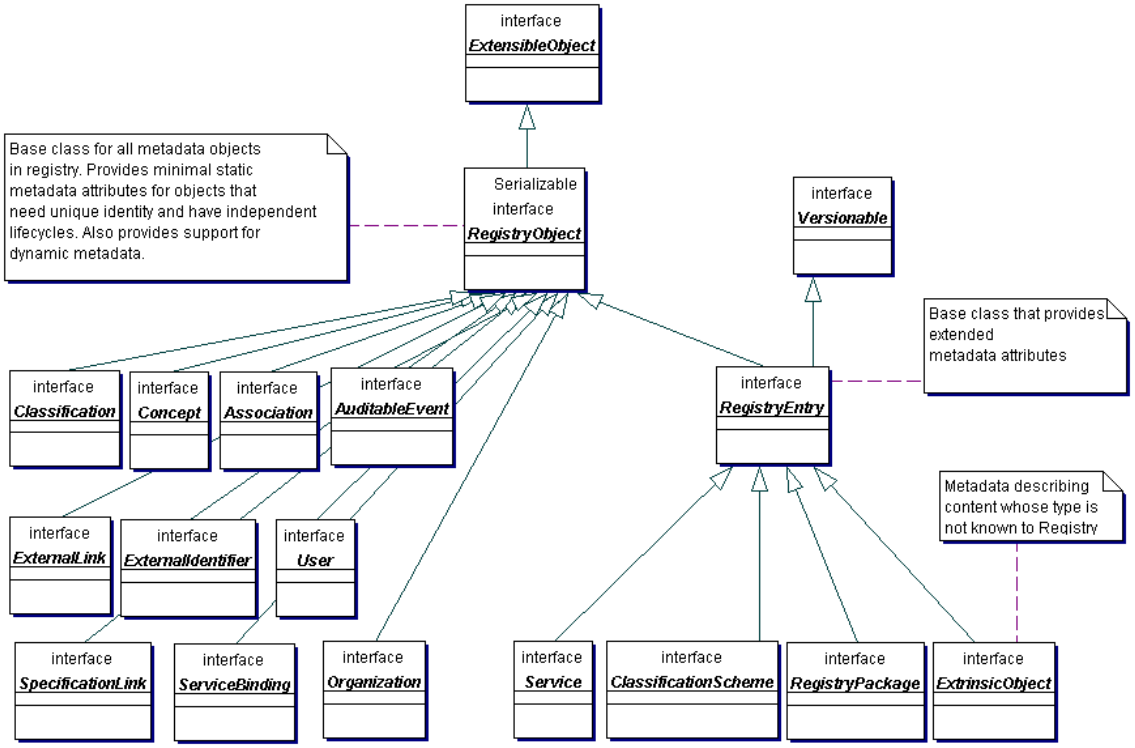


Figure 12: Information Model Inheritance View

4.2.1 RegistryEntry Interface

A few interfaces in the model represent high-level (coarse-grained) objects in the registry that require additional metadata such as version information and indication of the stability or volatility of the information.

The RegistryEntry interface is a base interface for the interfaces in the model that require additional metadata beyond what is provided by the relatively lighter-weight and more fine-grained RegistryObject interface.

4.2.2 ExtrinsicObject Interface

ExtrinsicObject instances provide metadata for a repository item (e.g. a WSDL document or an XML schema document) about which the registry has no prior knowledge. The ExtrinsicObject interface provides access to a repository item in the JAXR API.

An ExtrinsicObject instance is required for each repository item.

4.3 Internationalization (I18N) Support

Some information model classes have String attributes that are I18N capable and may be localized into multiple native languages. Examples include the name and description attributes of the RegistryObject interface as defined by the set/getName and set/getDescription methods of the RegistryObject interface.

The information model defines the InternationalString and the LocalizedString interfaces to support I18N capable attributes within the information model interfaces. These classes are defined below.

4.3.1 Interface InternationalString

This interface is used as a replacement for the String type whenever a String attribute needs to be I18N capable. An instance of the InternationalString interface composes a Collection of LocalizedString instances within it. Each LocalizedString instance provides a String value that is specific to a particular Locale and character set. The InternationalString interface provides set/get methods for adding or getting locale and character set specific String values for the InternationalString instance. Each LocalizedString within an InternationalString must have a unique Locale and character set name combination within that InternationalString.

4.3.2 Interface LocalizedString

This interface is used as a simple wrapper interface that associates a String with its Locale and character set. The interface is needed in the InternationalString interface where a Collection of LocalizedString instances are kept. Each LocalizedString instance has a Locale, a character set name and a String value.

4.4 Registry Audit Trail

This section describes the information model elements that support the audit trail capability of the registry.

The `getAuditTrail` method of a RegistryObject returns an ordered Collection of AuditableEvents. These AuditableEvents constitute the audit trail for the RegistryObject. AuditableEvents include a timestamp for the event. Each AuditableEvent has a reference to a User instance that identifies the specific user that performed the action that resulted in an AuditableEvent. Each User is affiliated with an Organization.

5 Classification of Registry Objects

This chapter describes in more detail, how the information model supports the classification of RegistryObjects. The ability to classify RegistryObjects is one of the most significant features provided by a registry. This is because classification facilitates rapid discovery of RegistryObjects within the registry.

5.1 Interface Classification

The Classification interface is used to classify RegistryObject instances. A RegistryObject may be classified along multiple dimensions by adding zero or more Classification instances to the RegistryObject. For example, an Organization may be classified by its industry, by the products it sells, by its geographical location and any other criteria. In this example, the RegistryObject would have at least three Classification instances added to it (industry, product and geography).

The RegistryObject interface provides several `addClassification` methods to allow a client to add Classification instances to a RegistryObject.

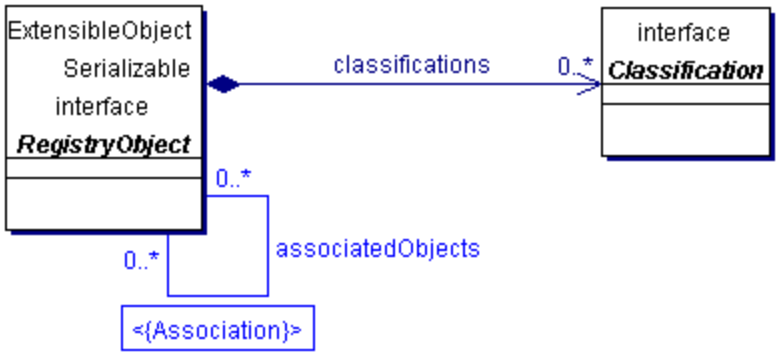


Figure 13: Classification of Registry Objects

Figure 13 shows how a RegistryObject may have zero or more Classification instances defined to classify it along multiple dimensions.

5.2 Interface ClassificationScheme

The ClassificationScheme interface is used to represent taxonomies that may be used to provide taxonomy values that can be used to classify or categorize RegistryObject instances.

[Note] The term taxonomy and ClassificationScheme are

```
            synonymous in this specification. Taxonomies
            are represented using a ClassificationScheme
            instance in the JAXR information model.
```

A Classification instance uses a ClassificationScheme instance to identify the taxonomy used to classify its RegistryObject. The ClassificationScheme instance provides the Classification with a taxonomy system that is used by the Classification. For example, a Geography ClassificationScheme can provide a taxonomy system that defines a geography structure with continents, countries within continents, states (or provinces or other internal subdivisions) within countries and cities and towns within states.

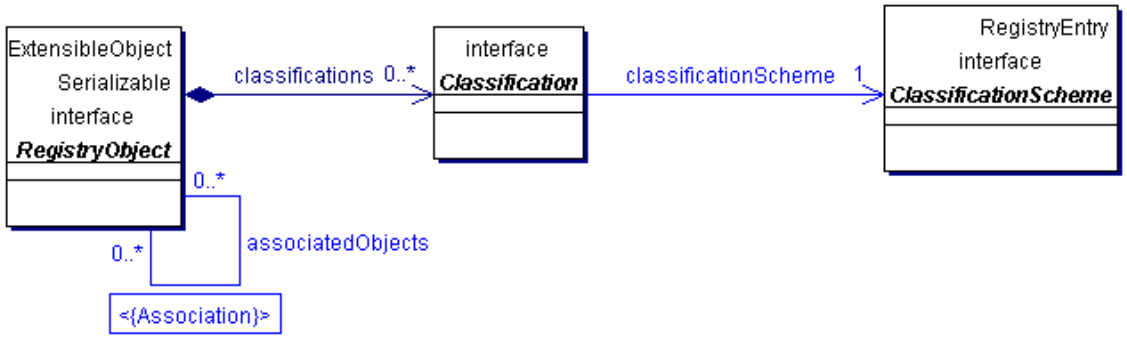


Figure 14: Role of ClassificationSchemes in Classification

Figure 14 shows how a Classification is associated with exactly one ClassificationScheme instance to identify the taxonomy used to classify a RegistryObject.

5.3 Taxonomy Structure and Elements

A taxonomy must define its structure in terms of its constituent taxonomy elements and their relationship to each other. For example, in a *Geography* taxonomy the country elements are contained within continent elements as illustrated in Figure 15.

A Classification instance needs some way to identify a specific taxonomy element within a taxonomy, in order to classify a RegistryObject. While a Classification uses a ClassificationScheme to identify a taxonomy for the classification, it needs additional information to identify a specific taxonomy element within that taxonomy.

5.3.1 Internal Vs. External Taxonomies

A taxonomy may be represented within a JAXR provider in one of the following ways:

a. The taxonomy elements and their structural relationship with each other are available within the JAXR provider. This case is referred to as *Internal Taxonomy* since the structure of the taxonomy is available internally to the JAXR provider.

b. The taxonomy elements and their structural relationship with each other are represented somewhere external to the JAXR provider and are not available to it. This case is referred to as *External Taxonomy* since the structure of the taxonomy is not available to the JAXR provider.

Internal taxonomies provide more functionality or value to the client in the form of the ability to browse the taxonomy structure and to validate that references to a taxonomy element in a Classification are meaningful and correct. The downside of internal taxonomies is that someone needs to submit that taxonomy to the registry and to be its maintainer.

In contrast, the upside of external taxonomies is that they are more resilient to changes in the taxonomy. Once a ClassificationScheme is submitted, the client can use it immediately without having to import the complete structure of that taxonomy and to maintain it as the taxonomy structure evolves. The downside of external taxonomies is that they do not support the ability to browse the taxonomy or to validate that references to a taxonomy element in a Classification are meaningful and correct.

5.3.2 Internal Vs. External Classifications

The Classification interface allows the classification of RegistryObjects using a ClassificationScheme whether the ClassificationScheme represents an internal taxonomy or an external taxonomy. A Classification instance that uses a Concept within an internal ClassificationScheme is referred to as an internal Classification. A Classification instance that uses a value within an external ClassificationScheme, is referred to as an external Classification.

5.4 Interface Concept

The Concept interface is used to represent taxonomy elements and their structural relationship with each other in order to describe an internal taxonomy.

Concept instances are used to define tree structures where the root of the tree is a ClassificationScheme instance and each node in the tree is a Concept instance.

Two Concepts may be defined as equivalent, as described in Appendix B. This is useful in use cases where we need to create a mapping between two different information model elements. For example, Concept equivalence is used in mapping the attributes of the PostalAddress interface in the information model to a Concept in an internal postal address ClassificationScheme. This is defined in detail in D.6.1.3.

Figure 15 shows how Concept instances are used to represent taxonomy elements and their structural relationship with each other in order to describe an internal taxonomy.

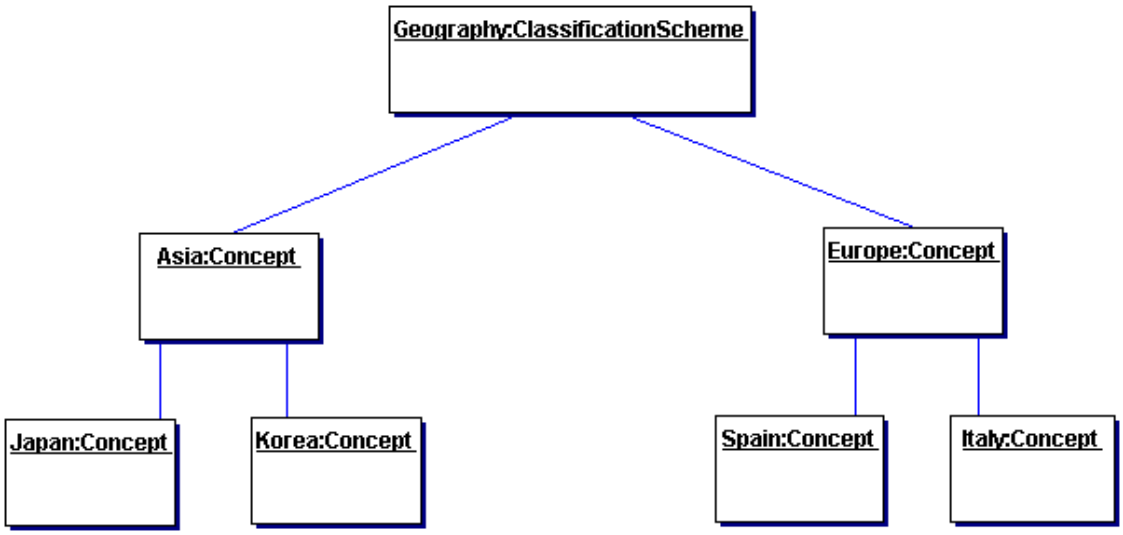


Figure 15: Role of Concepts in Representing Taxonomy Structure

5.5 Internal Classification

A Classification instance that is used to classify a RegistryObject using an internal taxonomy is referred to as an internal Classification. A client may call the `setConcept` method on a Classification and define a reference to a Concept instance from the Classification instance in order for that Classification to use an internal taxonomy. It is not necessary for the client to call setClassificationScheme for internal Classifications, because the classifying Concept already knows its root ClassificationScheme. For an internal classification, Classification.getName() must return the same value as Classification.getConcept().getName().

5.5.1 An Example of Internal Classification

Figure 16 shows an example of internal classification using a Concept to represent a taxonomy element. The example classifies an Organization instance as a Book Publisher using the NAICS standard taxonomy available as an internal taxonomy.

To save space, Figure 16 does not show all the Concepts between the “Book Publishers” node and the NAICS ClassificationScheme. Had they been there, they would have been linked together by the parent attribute of each Concept.

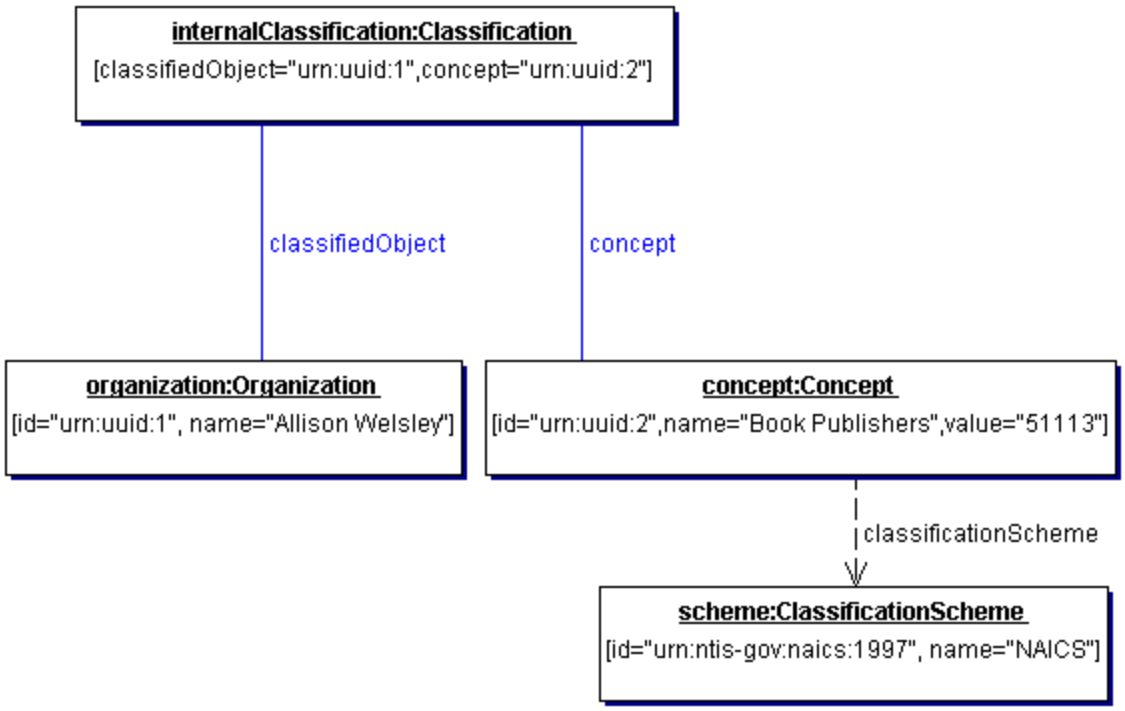


Figure 16: An Example of Internal Classification

5.6 External Classification

A Classification instance that is used to classify a RegistryObject using an external taxonomy is referred to as an external Classification. A client may call the `setValue` method on a Classification and define a unique value that logically represents a taxonomy element within the taxonomy whose structure is defined externally. A client may call the setClassificationScheme method for external Classifications to define the ClassificationScheme that represents the external taxonomy.

5.6.1 An Example of External Classification

Figure 17 shows an example of external classification. The example uses the same scenario, where a Classification classifies an Organization instance as a Book Publisher using the NAICS standard taxonomy. However, this time the structure of the NAICS taxonomy is not available internally to the registry, and consequently there is no Concept instance. Instead, the name and value attributes of the Classification are used to identify the Book Publishers taxonomy element. Note that name is optional but value is required.

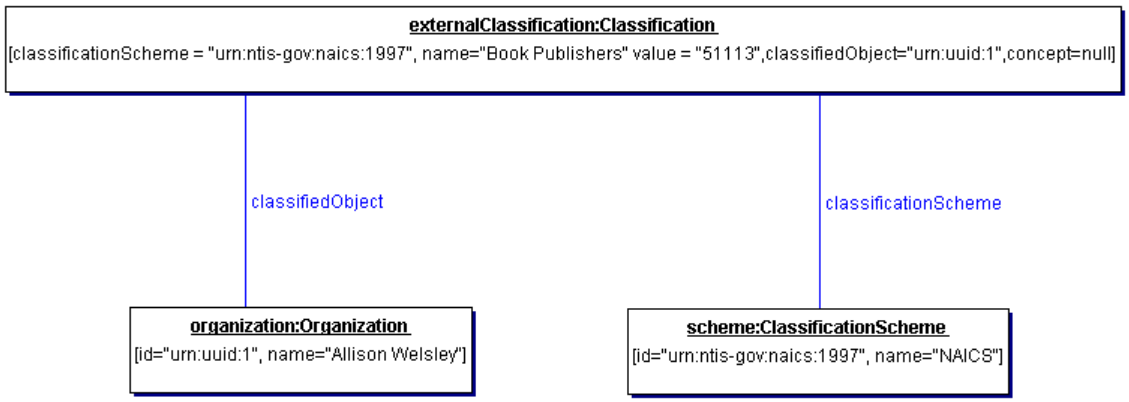


Figure 17: An Example of External Classification

5.7 An Example of Multiple Classifications

The next example shows how a RegistryObject may be classified by multiple classification schemes. In this example, two internal ClassificationSchemes named Industry and Geography are used to classify several Organization RegistryObjects by their industry and geography.

In Figure 18, in order to save space and improve readability, the Classification instances are not explicitly shown but are implied as associations between the RegistryObjects (shaded leaf node) and the associated Concepts.

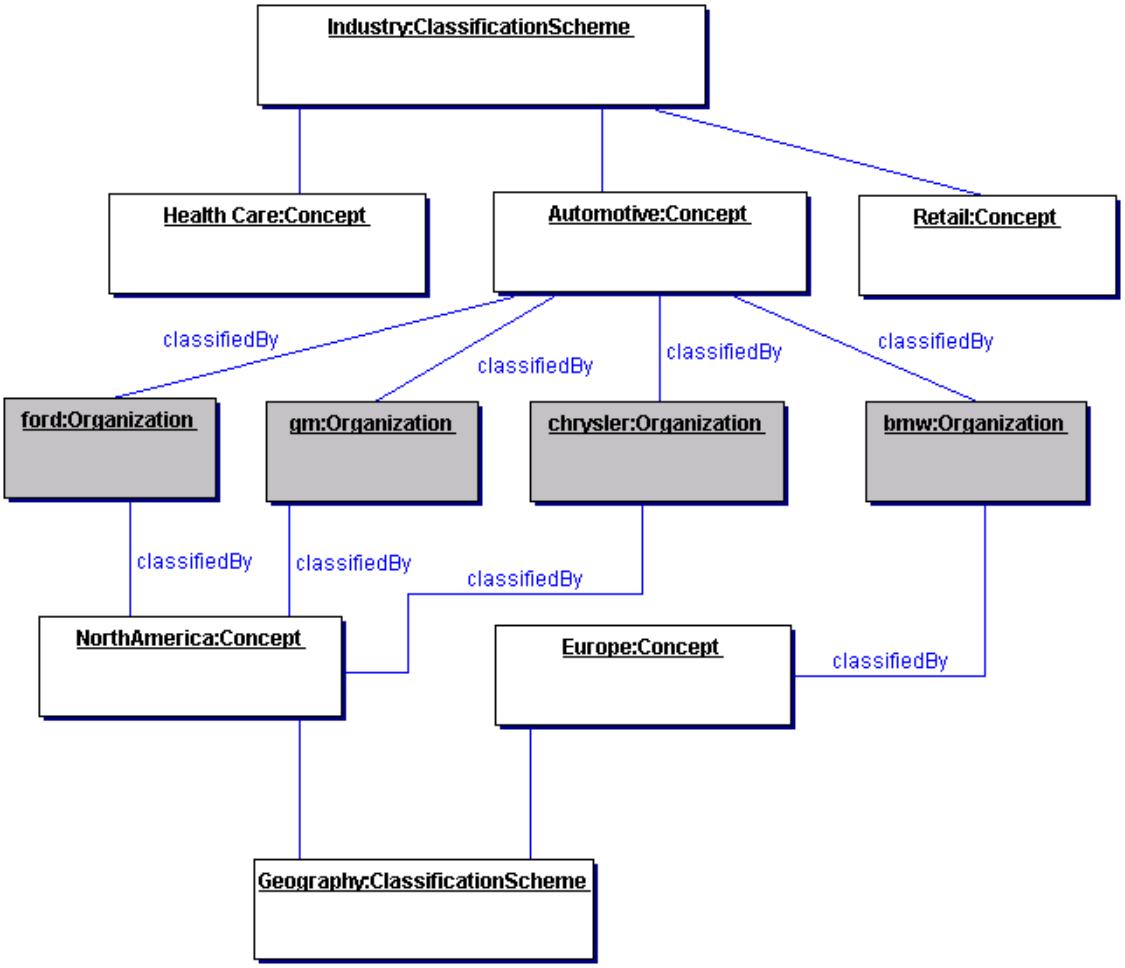


Figure 18: An Example of Multiple Classifications

[Note] It is important to point out that the dark nodes are not part of the Concept tree. The leaf nodes of the Concept tree are *Health Care, Automotive, Retail, NorthAmerica and Europe*. The dark nodes are associated with the Concept tree via a Classification instance that is not shown in the figure.

5.8 Context-sensitive Classification

[Note] The contents of this section are for illustrative purposes only.

Consider the case depicted in Figure 19, where a Collaboration Protocol Profile for ACME Inc. is classified by the *Japan* Concept under the Geography classification scheme. In the absence of the context for this classification, its meaning is ambiguous. Does it mean that ACME is located in Japan, or does it mean that ACME ships products to Japan, or does it have some other meaning? To address this ambiguity, a Classification may optionally be classified by another Concept (in this example named isLocatedIn) that provides the missing context for the Classification. Another Collaboration Protocol Profile for MyParcelService may be classified by the same *Japan* Concept, where this Classification is associated with a different Concept (in this example named shipsTo) to indicate a different context from the one used by ACME Inc.

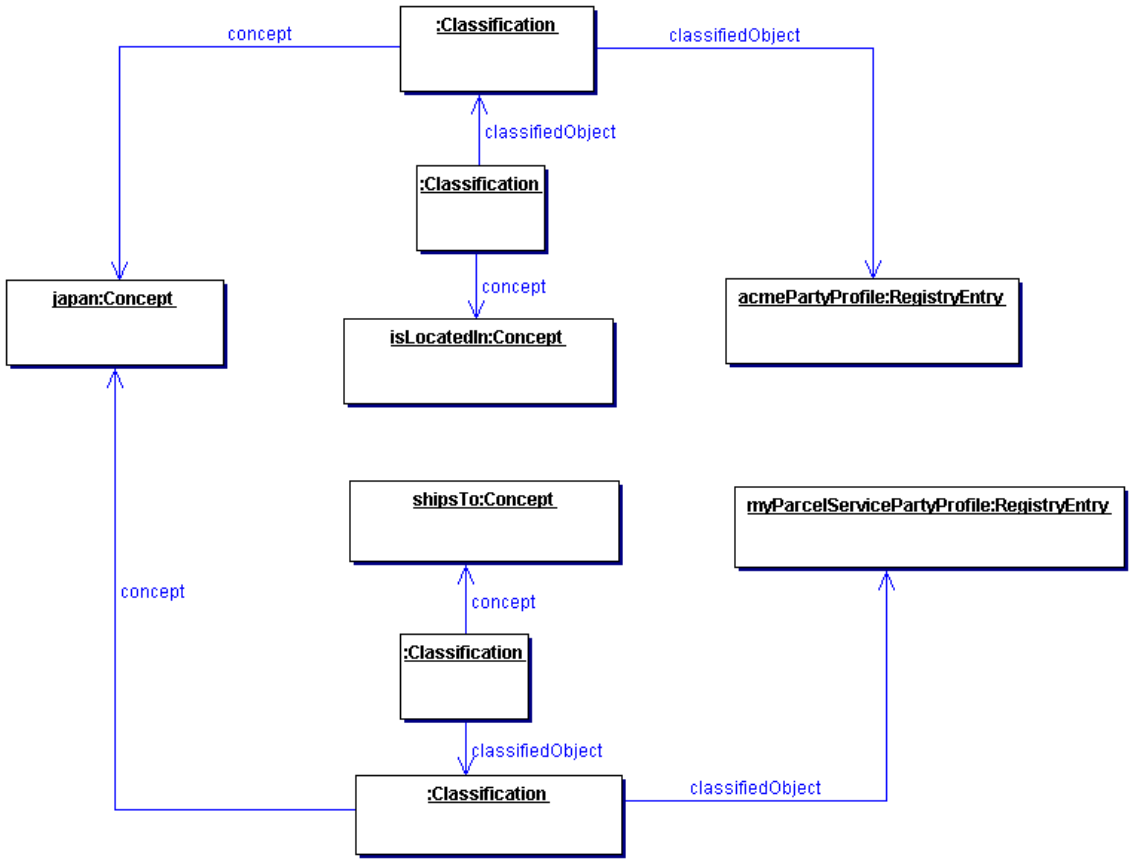


Figure 19: Context Sensitive Classification

Thus, in order to support the possibility of Classification within multiple contexts, a Classification may itself be classified by any number of Classifications that bind the first Classification to Concepts that provide the missing contexts.

In summary, the generalized support for classification schemes in the information model allows a submitting organization to:

1. Classify a RegistryObject by submitting an internal or external Classification.
2. Classify a RegistryObject along multiple facets by submitting multiple classifications.
3. Qualify a classification submitted for a RegistryObject by the contexts in which it is being classified.

6 Association of Registry Objects

A RegistryObject instance may be *associated* with zero or more RegistryObject instances. The information model defines an Association interface, an instance of which may be used to associate any two RegistryObject instances.

6.1 Example of an Association

One example of such an association is between two ClassificationScheme instances, where one ClassificationScheme supersedes the other ClassificationScheme, as shown in Figure 20. This may be the case when a new version of a ClassificationScheme is submitted.

In Figure 20, we see how an Association is defined between a new version of the NAICS ClassificationScheme and an older version of the NAICS ClassificationScheme.

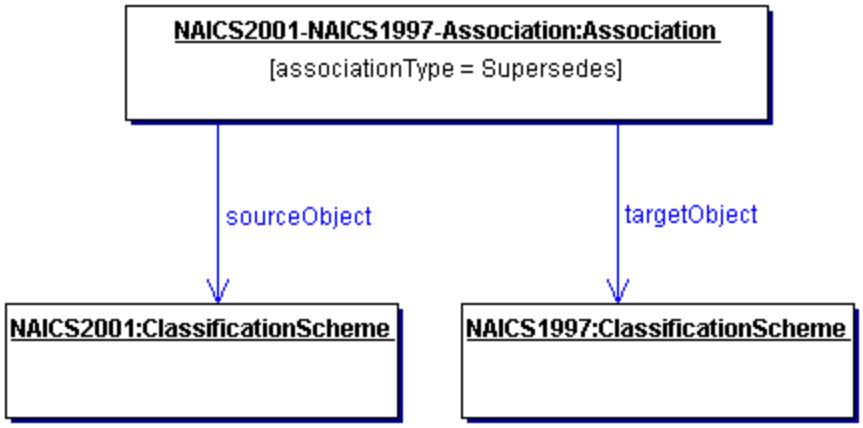


Figure 20: Example of RegistryObject Association

6.2 Source and Target Objects

An Association instance represents an association between a *source* RegistryObject and a *target* RegistryObject. These are referred to as the *sourceObject* and *targetObject* for the Association instance. It is important which object is the sourceObject and which is the targetObject, because it determines the directional semantics of an Association.

1303 In the example in Figure 20, it is important to make the newer version of NAICS
1304 ClassificationScheme be the sourceObject and the older version of NAICS be the
1305 targetObject because the associationType implies that the sourceObject
1306 supersedes the targetObject (and not the other way around).

1307 6.3 Association Types

1308 Each Association must have an associationType attribute that identifies the type
1309 of that association. The associationType attribute is a reference to an
1310 enumeration Concept as defined by the extensible ClassificationScheme defined
1311 in A.4. Our example uses the predefined associationType Concept named
1312 *Supersedes*.

1313 6.4 Intramural Associations

1314 A common use case for the Association interface is when a User “u” creates an
1315 Association “a” between two RegistryObjects “o1” and “o2”, where association “a”
1316 and RegistryObjects “o1” and “o2” are objects that were created by the same
1317 User “u”. This is the simplest use case, where the association is between two
1318 objects that are owned by same User that is defining the Association. Such
1319 associations are referred to as *intramural associations*.

1320 Figure 21 below extends the previous example in Figure 20 for the intramural
1321 association case.

1322

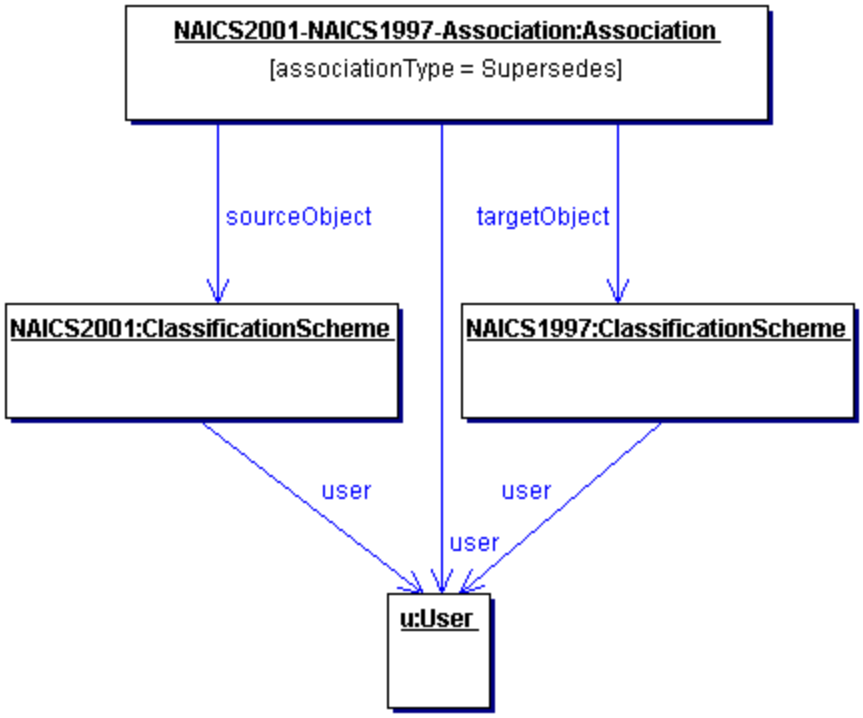


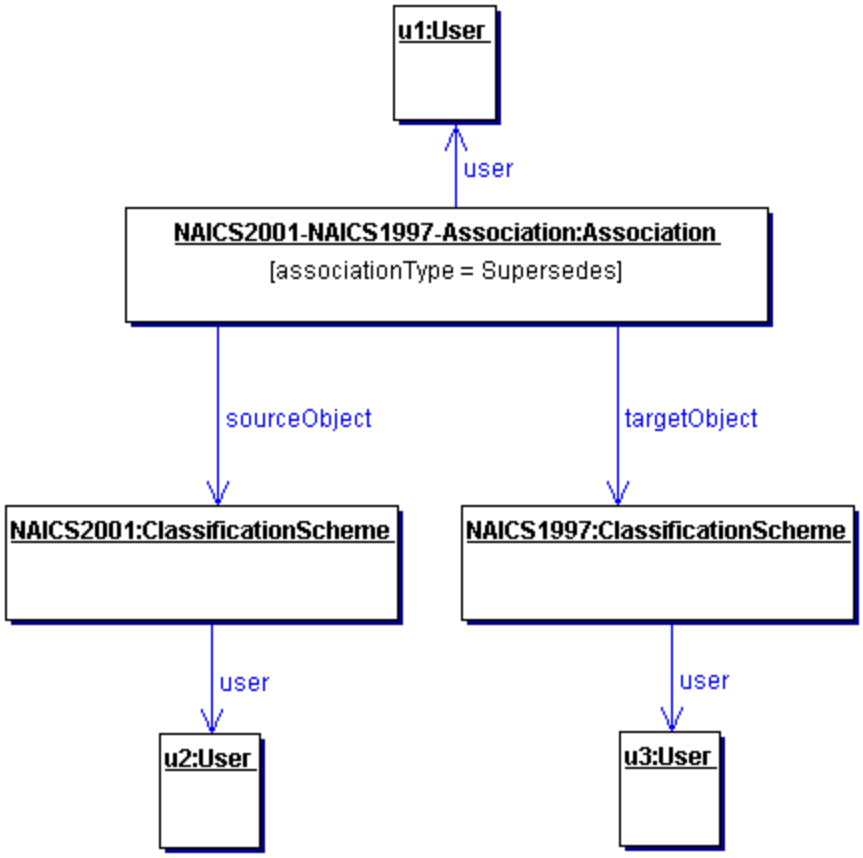
Figure 21: Example of Intramural Association

6.5 Extramural Association

The information model also allows a more sophisticated use case, where a User "u1" creates an Association "a" between two RegistryObjects "o1" and "o2", where association "a" is owned by User "u1", but RegistryObjects "o1" and "o2" are owned by User "u2" and User "u3" respectively.

In this use case the Association is being defined where either or both objects that are being associated are owned by a User different from the User defining the Association. Such associations are referred to as *extramural associations*. The Association interface provides a convenience method called `isExtramural` that returns true if the Association instance is an extramural Association.

Figure 22 extends the example in Figure 20 for the extramural association case. Note that it is possible for an extramural association to have two distinct Users rather than three distinct Users as shown in Figure 22. In such a case, one of the two users owns two of the three objects involved (Association, sourceObject, and targetObject).



1340
1341 **Figure 22: Example of Extramural Association**

1342 6.6 Confirmation of an Association

1343 An association may need to be confirmed by the parties whose objects are
1344 involved in that Association as the sourceObject or targetObject. This section
1345 describes the semantics of confirmation of an association by the parties involved.

1346 6.6.1 Confirmation of Intramural Associations

1347 Intramural associations may be viewed as declarations of truth and do not
1348 require any explicit steps to confirm that Association as being true. In other
1349 words, intramural associations are implicitly considered confirmed.

6.6.2 Confirmation of Extramural Associations

Extramural associations may be viewed as a unilateral assertion that may not be viewed as truth until it has been confirmed by the other (extramural) parties (Users “u2” and “u3” in example in section 6.5). The `confirmAssociation` method of the BusinessLifeCycleManager interface may be called by the extramural parties that own the sourceObject or targetObject to confirm an Association.

6.6.3 Undoing Confirmation of Extramural Associations

The `unConfirmAssociation` method on the BusinessLifeCycleManager interface may be called by the extramural parties that own the sourceObject or targetObject to undo a previous confirm on an Association.

6.7 Visibility of Unconfirmed Associations

Extramural associations require each extramural party to confirm the assertion being made by the extramural Association before the Association is visible to 3rd parties that are not involved in the Association. This ensures that unconfirmed Associations are not visible to 3rd party registry clients.

In order for a caller to find the Associations that it needs to confirm, it can use the `findCallerAssociations` method of the BusinessQueryManager interface.

6.8 Possible Confirmation States

Assume the most general case where there are three distinct User instances for an extramural Association. This case is illustrated in Figure 22. The extramural Association needs to be confirmed by both the other (extramural) parties (Users “u2” and “u3” in example) in order to be fully confirmed. The methods `isConfirmedBySourceOwner` and `isConfirmedByTargetOwner` in the Association interface provide access to the confirmation state for both the sourceObject and targetObject. A third convenience method, called `isConfirmed` provides a way to determine whether the Association is fully confirmed or not. So there are the following four possibilities related to the confirmation state of an extramural Association:

Confirmed By Owner of Source Object	Confirmed By Owner of Target Object	Comments
No	No	Unconfirmed
No	Yes	Unconfirmed (confirmed

		by target owner only).
Yes	No	Unconfirmed (confirmed by source owner only).
Yes	Yes	Confirmed.

1380

1381

7 Connection Management

This chapter specifies how a JAXR provider manages JAXR Connections. It provides details on the ConnectionFactory interface and the Connection interface.

7.1 Looking Up a ConnectionFactory

A JAXR `ConnectionFactory` object is configured in a provider-specific way to create connections with registry providers.

7.1.1 Looking Up a ConnectionFactory Using the JNDI API

The preferred way for a client to look up a JAXR ConnectionFactory is within the Java Naming and Directory Interface™ (JNDI) API.

A `ConnectionFactory` object is registered with a naming service in a provider specific way, such as one based on the JNDI API. This registration associates the `ConnectionFactory` object with a logical name. When an application wants to establish a connection with the provider associated with that `ConnectionFactory` object, it does a lookup, providing the logical name. The application can then use the `ConnectionFactory` object that is returned to create a connection to the messaging provider.

7.1.2 Looking Up a ConnectionFactory Without Using the JNDI API

The JAXR API provides an alternative way to look up a JAXR ConnectionFactory that does not require the use of the JNDI API. This is done using the newInstance static method on the abstract class ConnectionFactory provided in the JAXR API. The newInstance method returns a JAXR ConnectionFactory. The client may indicate which factory class should be instantiated by the newInstance method by defining the system property `javax.xml.registry.ConnectionFactoryClass`.

If this property is not set, the JAXR provider must return a default ConnectionFactory instance.

7.2 Setting Connection Properties on ConnectionFactory

Once a ConnectionFactory is available to the client, the client may configure the ConnectionFactory with a Properties object by calling the setProperties method on ConnectionFactory.

The properties specified may be either *standard properties* or *provider-specific properties*.

1. Standard properties are defined by the JAXR API.
2. Provider-specific properties are defined by a specific JAXR provider.

Connection configuration properties must be qualified by a package name. Standard connection configuration properties use the package `javax.xml.registry`, while provider-specific properties use the top-level package name for the provider (e.g. `com.sun.xml.registry.ebxml`).

7.2.1.1 Standard Connection Properties

The following table describes those connection properties that are standardized by this specification. JAXR providers may define additional properties specific to that provider.

Property	Data type	Description
`javax.xml.registry.queryManagerURL`	String	URL to the query manager service within the target registry provider.
`javax.xml.registry.lifeCycleManagerURL`	String	URL to the life cycle manager service within the target registry provider. If un-specified, must default to value of the `queryManagerURL` described above.
`javax.xml.registry.semanticEquivalences`	String	Allows specification of semantic equivalences as described in Appendix B.
`javax.xml.registry.security.authenticationMethod`	String	Provides a hint to the JAXR provider on the authentication method to be used when authenticating with the registry provider.
`javax.xml.registry.uddi.maxRows`	Integer	Specifies the maximum number of rows to be returned for find operations. This property is specific for UDDI providers.
`javax.xml.registry.postalAddressScheme`	String	Specifies the id of a ClassificationScheme that is used as the default postal address scheme for this connection. See D.6.1.3 for details.

7.3 Creating a JAXR Connection

To create a Connection to a registry provider, a client uses the `createConnection` method of a JAXR provider's `ConnectionFactory`.

public Connection **createConnection**()
throws JAXRException

The createConnection method must check that the `javax.xml.registry.queryManagerURL` described above is defined. If it is not defined, then the method must throw an InvalidRequestException. The createConnection method may also check if the URL specified is a valid URL.

7.4 Synchronous Connections

The JAXR client uses the `setSynchronous` method on a Connection to dynamically alter whether it receives responses and exceptions from the JAXR provider synchronously or not. The JAXR provider must use this communication preference when processing requests on behalf of that client.

If the communication preference is synchronous, the JAXR provider must process each request method call completely in a synchronous manner before it returns a non-null JAXRResponse (or a sub-interface) instance that contains the response to the request. The client thread must block until the JAXR provider has synchronously processed the request. The processing usually involves a round-trip interaction with a registry provider.

7.5 Asynchronous Connections

The JAXR client may indicate an asynchronous communication mode by calling the `setSynchronous` method on a Connection with a `false` parameter.

If the communication preference is asynchronous, each request method call returns a non-null JAXRResponse (or a sub-interface) immediately. The JAXR provider may spawn a separate thread to process the client request asynchronously.

[Note] In this version of the specification, asynchronous communication mode is not required within a J2EE container environment. The reason is that asynchronous mode support typically requires threads, sockets and so on, which are not allowed within a J2EE component.

7.5.1 JAXRResponse and Futures Design Pattern

The JAXRResponse returned immediately by the JAXR provider may not be immediately available. It uses a “futures” design pattern [Futures1, Futures2].

For an asynchronous request, a JAXR Provider will immediately give the client back a JAXRResponse instance even though its value is not available. Internally, the JAXR Provider will allocate a globally unique request ID. It is suggested that this ID be a DCE 128 bit UUID. This request ID is available to the client via the `getRequestId` method on JAXRResponse. The JAXR Provider must maintain the mapping between request IDs and JAXRResponse instances so that when a reply arrives from the registry provider at some time in the future, it can find the corresponding JAXRResponse instance and deliver the reply to the instance. This causes the value in the JAXRResponse instance to become available.

If a client attempts to read a value from a JAXRResponse that is not yet available, the JAXR provider must cause the caller to be blocked. Alternatively, a caller may examine a JAXRReponse for the availability of its value before attempting to read its value (and potentially blocking). The availability of the value may be polled by `getStatus` method, which must return STATUS_UNAVAILABLE when invoked on a JAXRResponse with a undefined value. In addition an `isAvailable` method on JAXRResponse is also provided as a convenience. The `isAvailable` method on JAXRResponse returns true or false depending upon whether the value is available or not.

Having a reference to a JAXRResponse does not block the client thread. The client thread is blocked only when it tries to access the reply contained in a JAXRResponse and the reply is not available.

7.6 Security Credentials Specification

The JAXR client uses the `setCredentials` method to dynamically alter its security credentials. These credentials provide details on the security-related identity associated with the client. An example of a credential is a username and password combination. The JAXR provider must use the credentials defined in the Connection instance at any given time when processing client requests. This may require having to re-authenticate with the registry provider in response to the `setCredentials` call when appropriate. The `setCredentials` method is described in more detail in section 10.5.1 within the context of JAXR security features.

7.7 Federated Connections

The JAXR API defines an interface named FederatedConnection. The FederatedConnection interface is a specialized sub-interface of the Connection interface. The FederatedConnection interface defines a single logical connection to multiple registry providers. A FederatedConnection is used in performing distributed or federated queries against target registry providers while treating them as a single logical registry provider. Federated queries are described in Section 9.7. The federated connection capability is optional in this version of the specification.

7.7.1 Creating a FederatedConnection

To create a federated connection to multiple registry providers, a client uses the `createFederatedConnection` method of a JAXR provider's `ConnectionFactory`.

public FederatedConnection **createFederatedConnection**(Collection connections) throws JAXRException

7.7.1.1 Connection Configuration

The client must specify a Collection of Connection instances to the createFederatedConnection method. These connections may be primitive connections or federated connections.

7.8 Concurrent Connections

A single JAXR client may concurrently maintain multiple connections. Each non-federated connection uses a single JAXR provider to access a single registry provider.

Collectively, these connections may concurrently access multiple registry providers.

Connection implementations must be thread-safe implementations.

7.9 Using a Connection to Access the Registry

Once a JAXR client has created a Connection using a ConnectionFactory, it can then use the Connection to access various capability specific interfaces. For example, it may use the Connection to:

a. Access the lifecycle management functionality of the JAXR provider to create, update and delete objects in the target registry provider.

1528 b. Access the query management functionality of the JAXR provider to find
1529 and retrieve objects from the target registry provider.

1530 The client must first get access to a `RegistryService` interface by calling the
1531 `getRegistryService` method on the Connection. The client may then call
1532 appropriate methods on the `RegistryService` interface to get references to
1533 various capability specific interfaces. For example, it may call the
1534 getBusinessLifeCycleManager method to get the
1535 `BusinessLifeCycleManager` interface for that Connection.

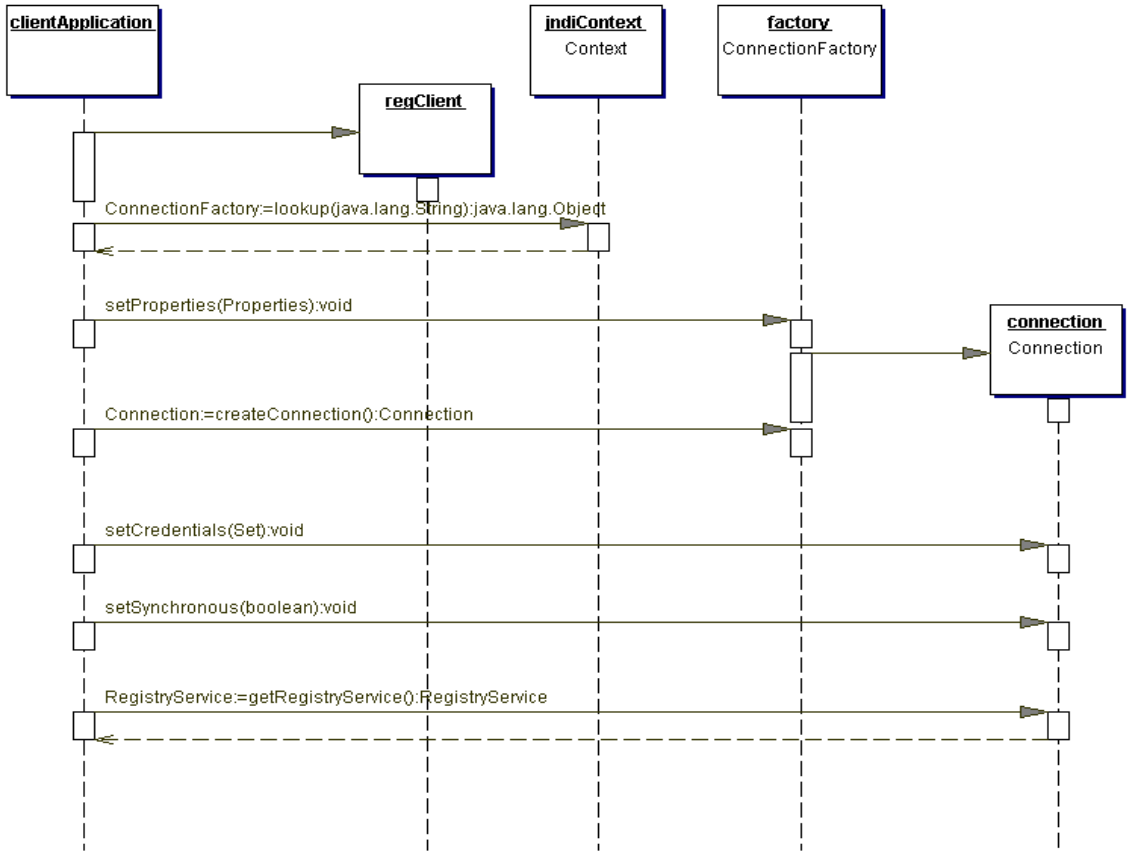
1536 7.10 Closing a Connection

1537 A JAXR provider typically allocates significant resources outside the JVM on
1538 behalf of a Connection. These resources include a network connection between
1539 the JAXR provider and the target registry provider shown at the bottom of Figure
1540 4. The network connection between a JAXR provider and a target registry
1541 provider is represented in Figure 4 by the block arrows marked ebXML/SOAP
1542 etc. The technical details of such network connections are registry provider-
1543 specific and therefore outside the scope of this specification.

1544 In order to conserve system resources, clients should close Connections when
1545 they are no longer needed. A client closes a Connection by calling the `close`
1546 method on it.

1547 7.11 Connection Setup Sequence

1548 Figure 23 illustrates the sequence of events during a typical JAXR Connection
1549 establishment.



1550

1551 **Figure 23: Connection Setup Sequence**

7.11.1 Connection Creation Code Sample

```
import javax.xml.registry.*;

…….

//Add system property to define which provider specific ConnectionFactory to use
System.setProperty("javax.xml.registry.ConnectionFactoryClass",
          "com.sun.xml.registry.uddi.ConnectionFactoryImpl");

//Create ConnectionFactory using class specified in System property and static newInstance
//method.
ConnectionFactory   factory = ConnectionFactory.newInstance();

//Define connection configuration properties
Properties props = new Properties();
props.put("javax.xml.registry.queryManagerURL", "http://java.sun.com/uddi/inquiry");
props.put("javax.xml.registry.lifeCycleManagerURL", "http://java.sun.com/uddi/publish");

//Create the connection passing it the configuration properties
factory.setProperties(props);
Connection connection = factory.createConnection();

Set credentials = new HashSet();
….
connection.setCredentials(credentials);
connection.setSynchronous(false);

RegistryService rs = connection.getRegistryService();

//Now get one or more capability specific interfaces
BusinessLifeCycleManager lcm = rs.getBusinessLifeCycleManager();
```

8 Life Cycle Management

This chapter specifies those aspects of the JAXR API that deal with managing the life cycle of registry objects. Examples of life cycle management operations include the ability to create, update and deleted registry objects.

Life cycle management interfaces are designed for use by the *Submitting Organization* or the *publisher* of registry metadata and content.

[Note] Some life cycle management requests may be privileged operations and require authentication and authorization. Security aspects of life cycle management is discussed in Chapter 10.

The JAXR API provides the following interfaces for managing life cycle of registry objects:

- Interface LifeCycleManager provides complete support for all life cycle management needs using a generic API.
- Interface BusinessLifeCycleManager extends the LifeCycleManager interface and provides the most common life cycle management capabilities for the key business objects in the information model in an explicit API.

8.1 Unique Key Assignment

As specified in the information model, every RegistryObject in the registry has a unique key. This key is usually generated by the registry provider. Some registry providers, such as ebXML Registry providers, optionally allow the submitter to specify this unique key.

The JAXR API allows a client to specify a key for a RegistryObject when submitting it to the registry. The client-supplied key must be used as the key for the object within the target registry provider, if all of the following conditions are true:

- The client supplies a key.
- The registry provider supports client supplied keys
- The client-supplied key is in a format acceptable by the specification governing the target registry. For example ebXML Registry specification requires client-supplied keys to be UUID based URNs.

A JAXR provider must ignore client-supplied keys if any of the above conditions are not met. In this case, the registry provider is expected to generate the unique key.

8.2 Interface *LifeCycleManager*

This interface provides complete support for all life cycle management needs using a generic API.

8.2.1 Requests, Responses and Exception Handling

The LifeCycleManager interface provides several methods that allow clients to make lifecycle requests with the target registry. Examples include:

- deleteObjects
- deprecateObjects
- saveObjects
- unDeprecateObjects

Each of these methods returns a BulkResponse object. The BulkResponse contains the Collection of response objects and may contain a Collection of RegistryException instances in case the request resulted in errors within the target registry provider.

If the JAXR provider detects an error in a client request prior to contacting the target registry provider, then the JAXR provider must throw a JAXRException. The JAXR provider must not catch runtime exceptions such as NullPointerException, since they are indicative of a programming error in the JAXR provider or the JAXR client. Instead, the JAXR provider should catch errors in a client request before they lead to potential runtime exceptions (e.g. NullPointerException). In such cases, the JAXR provider must throw a JAXRException that provides a clear indication as to the error in the client request and how to fix the error.

However, if the error is detected by the target registry provider, then the RegistryException is included within the BulkResponse instance's Collection of RegistryExceptions.

8.2.2 Creating Objects Using Factory Methods

The `LifeCycleManager` interface has several factory methods that follow the naming pattern `create<interface>` where <interface> represents the name of an interface in the javax.xml.registry.infomodel package.

These factory methods are used by clients to create different types of objects defined by the information model.

There is also a generic factory method called `createObject`, which allows clients to create any type of information model object. This method is useful for tool vendors who create tools supporting the JAXR API.

8.2.3 Saving Objects

An object created using a factory method initially exists only in memory. It is not saved in a registry provider until it is saved explicitly. The LifeCycleManager interface provides a saveObjects method for this purpose.

```
public BulkResponse saveObjects(java.util.Collection objects)
        throws JAXRException
```

The saveObjects method is used to create or update metadata and content.

This method saves a heterogeneous Collection of objects that are instances of RegistryObject sub-interfaces. Each such object in the collection is stored into the registry. Implementations must traverse object references from the object being saved and save them implicitly.

8.2.3.1 Interface *BulkResponse*

The BulkResponse interface is returned by many methods in the API where the response needs to include a Collection of objects. The BulkResponse interface is described here in the context of save methods. It behaves similarly in other usage contexts.

Each save method returns a `BulkResponse` instance. The `BulkResponse` instance contains a Collection of keys that are accessed via the `getCollection` method. These keys are for those objects that were saved successfully. The `BulkResponse` may also contain a Collection of `SaveException` instances as described next.

8.2.3.2 Interface SaveException

In event of a partial success where only a subset of objects was saved successfully, the getStatus method of the BulkResponse must return JAXRResponse.STATUS_WARNING. In this case, a Collection of `SaveException` instances is included in the `BulkResponse` instance. The `SaveExceptions` provide information on each error that prevented some objects in the save method Collection from being saved successfully.

Note that the `SaveExceptions` are returned as part of the `BulkResponse` instead of being thrown, in order to allow the `BulkResponse` to be returned despite the exception.

8.2.3.3 Implicit Saving of Objects

A JAXR provider must traverse object references from the object being saved and save them implicitly. For example, if the client saves an Organization explicitly, then the JAXR provider must implicitly save any Classifications, Associations, ExternalIdentifiers, Services, ServiceBindings etc., that are reachable from the Organization being saved.

8.2.4 Updating Objects

An object that is created and subsequently saved to the target registry provider can be updated by modifying the object in memory. Such updated objects must be saved using save methods such as saveObjects in order to be updated in the registry. The LifeCycleManager interface does not provide update methods that are distinct and separate from save methods.

8.2.5 Deleting Objects

An object that is created and subsequently saved to the target registry provider may be deleted from the target registry using a delete method of LifeCycleManager such as deleteObjects.

```
public BulkResponse deleteObjects (java.util.Collection keys)
            throws JAXRException
```

The deleteObjects method is used to delete previously submitted RegistryObject instances.

This method specifies a uniform Collection of Key instances identifying pre-existing objects in the registry.

An attempt to remove a RegistryObject while it is still the target of references may result in an InvalidRequestException that is returned within the BulkResponse, if the registry provider enforces such deletion constraints.

8.2.5.1 Interface DeleteException

This exception is used in the event of failure or partial success during a delete operation. A Collection of instances of this exception is returned in the BulkResponse for a delete method invocation if delete exceptions are encountered.

8.2.6 Deprecating Objects

Deprecating an object is an alternative to deleting it. Deprecating an object marks it as obsolete and likely to be deleted sometime in the future.

An object that is created and subsequently saved to the target registry provider may be deprecated from the target registry using a deprecate method of LifeCycleManager such as deprecateObjects.

```
public BulkResponse deprecateObjects(java.util.Collection keys)
        throws JAXRException
```

The deprecateObjects method is used to deprecate previously submitted RegistryObject instances.

This method specifies a uniform Collection of Key instances identifying pre-existing objects in the registry. Deprecating an object marks it as becoming obsolete. A deprecated object may remain in the registry for some time before it is deleted.

Once an object is deprecated, the JAXR provider must not allow any new references (e.g. new Associations, Classifications and ExternalLinks) to that object to be submitted. If a client makes an API call that results in a new reference to a deprecated object, the JAXR provider must throw a java.lang.IllegalStateException within a JAXRException. However, existing references to a deprecated object continue to function normally.

8.2.7 Undeprecating Objects

A deprecated object may be undeprecated using the unDeprecateObjects method of LifeCycleManager.

```
public BulkResponse unDeprecateObjects(java.util.Collection keys)
        throws JAXRException
```

1747 The unDeprecateObjects method is used to un-deprecate previously deprecated
1748 RegistryObject instances.

1749 This method specifies a uniform Collection of Key instances identifying pre-
1750 existing deprecated objects in the registry.

1751 Once an object is undeprecated, the JAXR provider must again allow new
1752 references (e.g. new Associations, Classifications and ExternalLinks) to that
1753 object to be submitted.

1754 8.3 Interface *BusinessLifeCycleManager*

1755 Interface `BusinessLifeCycleManager` defines a simple business-level API
1756 for life cycle management of some important high-level interfaces in the
1757 information model. This interface provides no new functionality beyond that of
1758 LifeCycleManager. The goal of defining this interface is to provide an API similar
1759 to that of the publisher's API in UDDI. The intent is to provide a familiar API to
1760 UDDI developers.

1761 The BusinessLifeCycleManager interface provides the ability to explicitly submit,
1762 update or delete instances of the most important high-level interfaces in the
1763 information model. These high-level interfaces include:

1764 - Interface `Organization`
1765 - Interface `Service`
1766 - Interface `ServiceBinding`
1767 - Interface `Concept`

1768

1769

Method Summary	
void	**confirmAssociation**(Association assoc) Confirms this Association by the User associated with the caller.
BulkResponse	**deleteAssociations**(java.util.Collection associationKeys) Deletes the Associations corresponding to the specified Keys.
BulkResponse	**deleteClassificationSchemes**(java.util.Collection schemeKeys) Deletes the ClassificationSchemes corresponding to the specified Keys.

BulkResponse	**deleteConcepts**(java.util.Collection conceptKeys) Deletes the Concepts corresponding to the specified Keys.
BulkResponse	**deleteOrganizations**(java.util.Collection organizationKeys) Deletes the organizations corresponding to the specified Keys.
BulkResponse	**deleteServiceBindings**(java.util.Collection bindingKeys) Deletes the ServiceBindings corresponding to the specified Keys.
BulkResponse	**deleteServices**(java.util.Collection serviceKeys) Deletes the services corresponding to the specified Keys.
BulkResponse	**saveAssociations**(java.util.Collection associations, boolean replace) Saves the specified Association instances.
BulkResponse	**saveClassificationSchemes**(java.util.Collection schemes) Saves the specified ClassificationScheme instances.
BulkResponse	**saveConcepts**(java.util.Collection concepts) Saves the specified Concepts.
BulkResponse	**saveOrganizations**(java.util.Collection organizations) Saves the specified Organizations.
BulkResponse	**saveServiceBindings**(java.util.Collection bindings) Saves the specified ServiceBindings.
BulkResponse	**saveServices**(java.util.Collection services) Saves the specified Services.
void	**unConfirmAssociation**(Association assoc) Undoes a previous confirmation of this Association by the User associated with the caller.

1770

8.3.1 Save Methods

The `BusinessLifeCycleManager` interface defines a set of save methods, one for each key interface (e.g. saveOrganizations). Each save method takes a Collection as parameter so it can save multiple objects of the type associated with the save method. That Collection contains objects that are instances of the type associated with the save method. For example, the `saveOrganizations` method accepts a Collection of Organization instances. If the Collection contains an object whose type does not match the `save` method, the implementation must throw an UnexpectedObjectException.

Note that a client must save an object using the appropriate save method when a setter method on the object is called by the client. Calling the setter method without saving the object will not save the modified object to the target registry.

The save methods of BusinessQueryManager are a convenience for those who prefer an explicit API. Calling a save method of BusinessQueryManager should have the same effect as calling the saveObjects method in LifeCycleManager.

8.3.2 Delete Methods

The `BusinessLifeCycleManager` interface defines a set of `delete` methods, one for each key interface (e.g. deleteOrganizations etc.). Each `delete` method takes a Collection as parameter so it can delete multiple objects of the type associated with the `delete` method. The Collection is homogeneous and contains keys to objects that are being deleted. For example, the `deleteOrganizations` method accepts a Collection of Key instances where each Key is the Key for an Organization instance. If the Collection contains a key whose object type does not match the `delete` method, the implementation must throw an UnexpectedObjectException.

The delete methods of BusinessQueryManager are a convenience for those who prefer an explicit API. Calling a delete method of BusinessQueryManager should have the same effect as calling the deleteObjects method in LifeCycleManager.

8.4 Life Cycle Management and Federated Connections

Life cycle management operations are not supported by federated connections as represented by a FederatedConnection. The `getLifeCycleManager` and `getBusinessLifeCycleManager` methods of `RegistryService` from a FederatedConnection must throw `UnsupportedCapabilityException`.

9 Query Management

This section specifies those aspects of the JAXR API that deal with querying the registry for registry object (metadata) and repository item (content) instances. Query management interfaces are designed for use by any non-privileged registry user, typically through a specialized JAXR client, such as a Registry Browser tool or an intelligent web agent (digital assistant).

[Note] `Security aspects of query management are discussed in Chapter 10.`

The API provides a common `QueryManager` interface as well as two specialized sub-interfaces:

- Interface `BusinessQueryManager`
- Interface `DeclarativeQueryManager`

9.1 Interface *QueryManager*

Interface `QueryManager` provides a common base class for all other specialized QueryManager sub-classes in the API. It has the following methods:

Method Summary	
`RegistryObject`	**`getRegistryObject`**`(java.lang.String id)` Gets the RegistryObject specified by the Id.
`RegistryObject`	**`getRegistryObject`**`(java.lang.String id, java.lang.String objectType)` Gets the RegistryObject specified by the Id and type of object.
`BulkResponse`	**`getRegistryObjects`**`()` Gets the RegistryObjects owned by the caller.
`BulkResponse`	**`getRegistryObjects`**`(java.util.Collection objectKeys)` Gets the specified RegistryObjects.
`BulkResponse`	**`getRegistryObjects`**`(java.util.Collection objectKeys, java.lang.String objectTypes)` Gets the specified RegistryObjects.

BulkResponse	**getRegistryObjects**(java.lang.String objectType) Gets the RegistryObjects owned by the caller, that are of the specified type.
RegistryService	**getRegistryService**() Returns the parent RegistryService that created this object.

9.2 Interface *BusinessQueryManager*

Interface `BusinessQueryManager` provides a simple business-level API that provides the ability to query for the most important high-level interfaces in the information model.

Many of the methods in this interface take similar arguments and have the same return type:

findQualifiers - a Collection of find qualifiers as defined by the FindQualifier interface. It specifies qualifiers that effect string matching, sorting, and boolean predicate logic and so on.

namePatterns - a Collection that may consist of either String or LocalizedString objects. Each String, or value within a LocalizedString, is a partial or full name pattern with wildcard searching as specified by the SQL-92 LIKE specification. Unless otherwise specified in findQualifiers, this is a logical OR and a match on any name qualifies as a match for this criteria.

classifications - a Collection of Classifications that classify the object. It is analogous to a catgegoryBag in UDDI. Unless otherwise specified in findQualifiers, this is a logical AND and requires a match on ALL specified Classifications to qualify as a match for this criteria. A transient Classification may be created by the programmer using LifeCycleManager.createClassification to use in this Collection.

specifications - a Collection of RegistryObjects that represent (proxy) a technical specification. It is analogous to a tModelBag in UDDI. Unless otherwise specified in findQualifiers, this is a logical AND and requires a match on ALL specified Specifications to qualify as a match for this criteria.

externalIdentifiers - a Collection of ExternalIdentifiers that provide an external identifier for the object using an identification scheme such as DUNS. It is analogous to an identifierBag in UDDI. Unless otherwise specified in findQualifiers, this is a logical AND and requires a match on ALL specified

Classifications to qualify as a match for this criteria. A transient ExternalIdentifier may be created by the programmer using LifeCycleManager.createExternalIdentifier to use in this Collection.

externalLinks - Ia Collection of ExternalLinks that link the object to content outside the registry. It is analogous to an overviewDoc in UDDI. Unless otherwise specified in findQualifiers, this is a logical AND and requires a match on ALL specified ExternalLinks to qualify as a match for this criteria.

BulkResponse - Contains Collection of objects returned by the find methods.

Method Summary

BulkResponse	**findAssociations**(java.util.Collection findQualifiers, java.lang.String sourceObjectId, java.lang.String targetObjectId, java.util.Collection associationTypes) Finds all Associations that match ALL of the criteria specified by the parameters of this call.
BulkResponse	**findCallerAssociations**(java.util.Collection findQualifiers, java.lang.Boolean confirmedByCaller, java.lang.Boolean confirmedByOtherParty, java.util.Collection associationTypes) Finds all Associations owned by the caller that match ALL of the criteria specified by the parameters of this call.
ClassificationScheme	**findClassificationSchemeByName**(java.util.Collection findQualifiers, java.lang.String namePattern) Find a ClassificationScheme by name based on the specified name pattern.
BulkResponse	**findClassificationSchemes**(java.util.Collection findQualifiers, java.util.Collection namePatterns, java.util.Collection classifications, java.util.Collection externalLinks) Finds all ClassificationSchemes that match ALL of the criteria specified by the parameters of this call.
Concept	**findConceptByPath**(java.lang.String path) Find a Concept based on the path specified.
BulkResponse	**findConcepts**(java.util.Collection findQualifiers,

	java.util.Collection namePatterns, java.util.Collection classifications, java.util.Collection externalIdentifiers, java.util.Collection externalLinks) Finds all Concepts that match ALL of the criteria specified by the parameters of this call.
BulkResponse	**findOrganizations**(java.util.Collection findQualifiers, java.util.Collection namePatterns, java.util.Collection classifications, java.util.Collection specifications, java.util.Collection externalIdentifiers, java.util.Collection externalLinks) Finds all Organizations that match ALL of the criteria specified by the parameters of this call.
BulkResponse	**findRegistryPackages**(java.util.Collection findQualifiers, java.util.Collection namePatterns, java.util.Collection classifications, java.util.Collection externalLinks) Finds all RegistryPackages that match ALL of the criteria specified by the parameters of this call.
BulkResponse	**findServiceBindings**(Key serviceKey, java.util.Collection findQualifiers, java.util.Collection classifications, java.util.Collection specifications) Finds all ServiceBindings that match ALL of the criteria specified by the parameters of this call.
BulkResponse	**findServices**(Key orgKey, java.util.Collection findQualifiers, java.util.Collection namePatterns, java.util.Collection classifications, java.util.Collection specifications) Finds all Services that match ALL of the criteria specified by the parameters of this call.

9.2.1 Find Methods

The `BusinessQueryManager` interface defines a set of find methods for each key interface. Most find methods can return multiple objects of the type associated with the find method that match the specified search criteria.

9.2.1.1 Collection Parameters

Any Collection parameter that is a Collection of objects is a homogeneous collection of objects of the appropriate type. If the Collection contains an object whose type is unexpected, the implementation must throw an UnexpectedObjectException.

Unless noted otherwise, all Collection parameters in the JAXR API have the following usage semantics. If the Collection is null, the JAXR provider must treat it as if it were an empty Collection. A JAXR provider may use null Collection values as a hint to optimize the processing of the Collection.

9.2.1.2 Interface FindException

This exception is used in the event of failure or partial success during a find operation. A Collection of instances of this exception is returned in the BulkResponse for a find method invocation if FindExceptions are encountered.

9.2.2 Canonical Paths Syntax for Concepts

In the `findConceptByPath` method, the desired Concept is indicated via a canonical representation that uniquely identifies the absolute path leading from the ClassificationScheme to that Concept.

The canonical path representation is defined by the following BNF grammar:

```
canonicalPath ::= '/' schemeId conceptPath
conceptPath  ::=      '/' conceptValue
                 |    '/' conceptValue ( conceptPath )?
```

In the above grammar, schemeId is the id attribute of the ClassificationScheme instance, and conceptValue is defined by NCName production as defined by http://www.w3.org/TR/REC-xml-names/#NT-NCName.

9.2.2.1 Example of Canonical Path Representation

The following canonical path represents the Concept with value ‘UnitedStates’ with a parent Concept with value ‘NorthAmerica’ under a ClassificationScheme with id ‘Geography-id’.

```
/Geography-id/NorthAmerica/UnitedStates
```

9.3 Interface *DeclarativeQueryManager*

Interface `DeclarativeQueryManager` provides a more flexible generic API that provides the ability to perform ad hoc queries using a declarative query language syntax. Currently the only declarative syntaxes supported are SQL-92 and OASIS ebXML Registry Filter Queries. Note that support of SQL queries is optional for OASIS ebXML Registries. If the target registry does not support SQL queries then methods calls on DeclarativeQueryManager should throw UnsupportedCapabilityException.

Method Summary	
Query	**createQuery**(int queryType, java.lang.String queryString) Creates a Query object given a queryType (for example, QUERY_TYPE_SQL) and a String that represents a query in the syntax appropriate for queryType.
BulkResponse	**executeQuery**(Query query) Executes a query as specified by query paramater.

9.3.1 Interface Query

The Query interface encapsulates a query in a declarative query language. Currently a Query can only be defined using an SQL-92 syntax or the OASIS ebXML Registry query syntax. In future support for other query languages such as XQuery may be added.

Method Summary	
int	**getType**() Gets the type of Query (e.g. SQL).
java.lang.String	**toString**() Must print the String representing the query.

9.3.2 Creating a Query

A JAXR client must first create a `Query` object to encapsulate its query in a supported declarative query syntax such as SQL. This is done by calling the `createQuery` factory method on the `DeclarativeQueryManager`. A JAXR provider may optionally perform client-side validation of the SQL query syntax and throw an `InvalidRequestException` when an invalid query is specified. In the absence of such validation, it is expected that the registry provider will detect the error, in which case a RegistryException will be returned in the BulkResponse.

9.3.3 Executing a Query

A JAXR client executes a declarative query encapsulated by a `Query` instance by invoking the `executeQuery` method of the `DeclarativeQueryManager`, giving it the `Query` object as parameter.

9.4 SQL Query Syntax

The syntax for the SQL Query is defined by a stylized use of a proper subset of the "SELECT" statement of Entry Level SQL defined by ISO/IEC 9075:1992, Database Language SQL [SQL], extended to include `sql invoked routines` (also known as stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM].

9.4.1 SQL Query Syntax Binding To Information Model

The SQL Queries are defined based upon the query syntax defined in [RIM] and a relational schema that is an algorithmic binding to the information model as described in the section titled "SQL Query Syntax Binding To [RIM]" in [RIM].

9.5 OASIS ebXML Registry Filter Query Syntax

The [ebRS] specification defines the filter query syntax for the OASIS ebXML Registry. This syntax is an XML syntax defined by an XML Schema.

9.6 Query Result

The executeQuery method returns a BulkResponse that contains a homogeneous collection of objects. The type of objects is defined by the FROM clause of the query. For example, `SELECT from Organization WHERE …` returns a Collection of Organization instances.

9.7 Federated Queries

A client may issue a federated query against multiple registry providers as if they were a single logical registry provider. A federated query is issued in a manner similar to a non-federated query by calling a method on either the `BusinessQueryManager` or `DelarativeQueryManager` interfaces. The only difference is that the client must use the `BusinessQueryManager` or `DelarativeQueryManager` interface that was obtained from a RegistryService of a FederatedConnection instance rather than of a primitive Connection. Federated query capability is an optional feature of a JAXR provider for version 1.0.

10 Security Architecture

The JAXR API does not specify its own security mechanisms. Instead, the API defines some minimal methods that allow implementations to choose their underlying security mechanism.

The minimal security-related methods in the JAXR API are aligned with the Java Authentication and Authorization Service (JAAS) and Java Secure Socket Extension (JSSE) specifications. These are a standard part of the Java 2 Platform, Standard Edition (J2SE™) version 1.4 software. Support for earlier versions of the J2SE software is available through stand-alone versions of these packages.

10.1 Integrity

To ensure the integrity of a JAXR request to the target registry provider, the JAXR API provides the ability for the request to the registry provider to be signed using a digital certificate. All Level 1 JAXR providers must be capable of sending signed requests to the registry provider and receiving signed responses from the registry provider.

The JAXR client does not directly sign requests, nor does it validate signed responses. Instead, this functionality is delegated to the Level 1 JAXR provider.

10.2 Confidentiality

To ensure the confidentiality of a JAXR request to the target registry provider, all JAXR providers (Level 0 and above) must be able to use SSL to communicate with a registry that is accessible over the HTTPS protocol. Use of the HTTPS protocol is transparent to the JAXR client except for the fact that the URL defines 'https' as the protocol.

Level 1 JAXR providers may be capable of sending encrypted requests to the registry provider and receiving encrypted responses from the registry provider.

The JAXR client does not directly encrypt requests, nor does it decrypt incoming encrypted responses from the registry. Instead, this functionality is delegated to the Level 1 JAXR provider.

10.3 Authentication

From the perspective of the target registry provider, it is the JAXR provider, not the JAXR client, that is the registry provider's client. The JAXR provider must authenticate with the registry provider as specified by the specification governing the registry provider (e.g. ebXML Registry, UDDI). Typically, such authentication occurs on certain privileged requests. For example, in UDDI, authentication is needed only on requests that use the UDDI publishing API to submit, update, or delete content.

In all cases, the JAXR provider initiates the authentication requests, while the registry provider performs the actual authentication.

The JAXR client does not directly initiate authentication. It does not need to know when authentication with the target registry is necessary nor how it must be done. Instead, this functionality is delegated to the JAXR provider.

10.3.1 Authentication Methods

A JAXR provider may support multiple authentication methods. A JAXR client may specify a particular authentication method using the `javax.xml.registry.security.authenticationMethod` connection property. If the provider does not support the specified authentication method then it must throw UnsupportedCapabilityException during the `ConnectionFactory.createConnection` call. The following authentication methods have been defined as normative authentication methods:

- UDDI_GET_AUTHTOKEN is the get_AuthToken protocol defined by [UDDI_API2].
- HTTP_BASIC is the HTTP basic authentication as specified in RFC2068.
- CLIENT_CERTIFICATE
- MS_PASSPORT is the Microsoft Passport authentication mechanism.

A provider may support one or more of these methods as well as additional provider-specific methods.

10.4 Authorization

The JAXR provider does not perform any authorization decisions. All requests from the client are relayed to the registry provider, along with any required authentication tokens. The registry provider may perform authorization checks in a registry provider-specific manner and report any authorization errors. The JAXR provider must map any such registry provider-specific errors to a JAXRException and deliver it to the client.

10.5 Security Support in JAXR API

The following section describes the support for security features in the JAXR API.

10.5.1 User Registration

The JAXR API does not provide any support for registration of User credentials with a registry provider. Such user registration is a one-time activity that must be performed out-of-band with respect to the JAXR API.

10.5.2 Method Connection.setCredentials

```
public void setCredentials(Set credentials)
                                        throws JAXRException
```

Allows the client to set the security credentials for the user that is currently associated with the client. The term credential has the meaning defined by the JAAS API. A credential may be any java.lang.Object instance that is supported as a credential type by the JAXR provider.

A JAXR provider must support credentials in the form of either a digital certificate and private key or a username and password combination. A JAXR provider may support other forms of security credentials. A JAXR provider is not required to support certain forms of credentials if they are not supported by the target registry provider. For example, if digital certificates are not supported by UDDI, then a JAXR UDDI provider need not support digital certificate credentials.

When a JAXR client specifies a digital certificate as a credential, it must do so using an instance of the javax.security.auth.x500.X500PrivateCredential class as defined by the JAAS API.

When a JAXR client specifies a username and password combination as a credential, it must do so using an instance of the java.net.PasswordAuthentication class. This class is a simple container for a username and password.

A JAXR provider must be able to use the appropriate credential from the credentials set for the Connection by the last setCredentials call and authenticate with the registry provider in a provider-specific manner.

If a client dynamically changes its credentials, the change has no impact on the pre-existing RegistryService instance within that Connection. Nor does it have any impact on any information model objects created within that Connection.

Appendix A Pre-defined Enumerations

This section defines the pre-defined enumerations used by the JAXR API. These enumerations are defined as Concept hierarchies (a ClassificationScheme and a set of child Concepts).

The enumerations are listed using the following notational convention. Each enumeration is a separate section within this appendix. The name of the ClassificationScheme of each enumeration is the name of the enumeration and is used as the section title.

A.1 Identification of Pre-defined Enumerations

A client may identify the ClassificationScheme for a pre-defined enumeration using the name of the ClassificationScheme in the en_US locale. Consequently, the ClassificationScheme for a pre-defined enumeration must always have a name defined in en_US locale. So to identify a Concept with value of “Service” within the pre-defined enumeration ObjectType in a findConceptByPath call, a client writes the following code:

```
Concept  serviceConcept = bqm.findConceptByPath(‘/ObjectType/Service’);
```

A.2 Enumeration ObjectType

The ObjectType enumeration is used in the getObjectType method of RegistryObject.

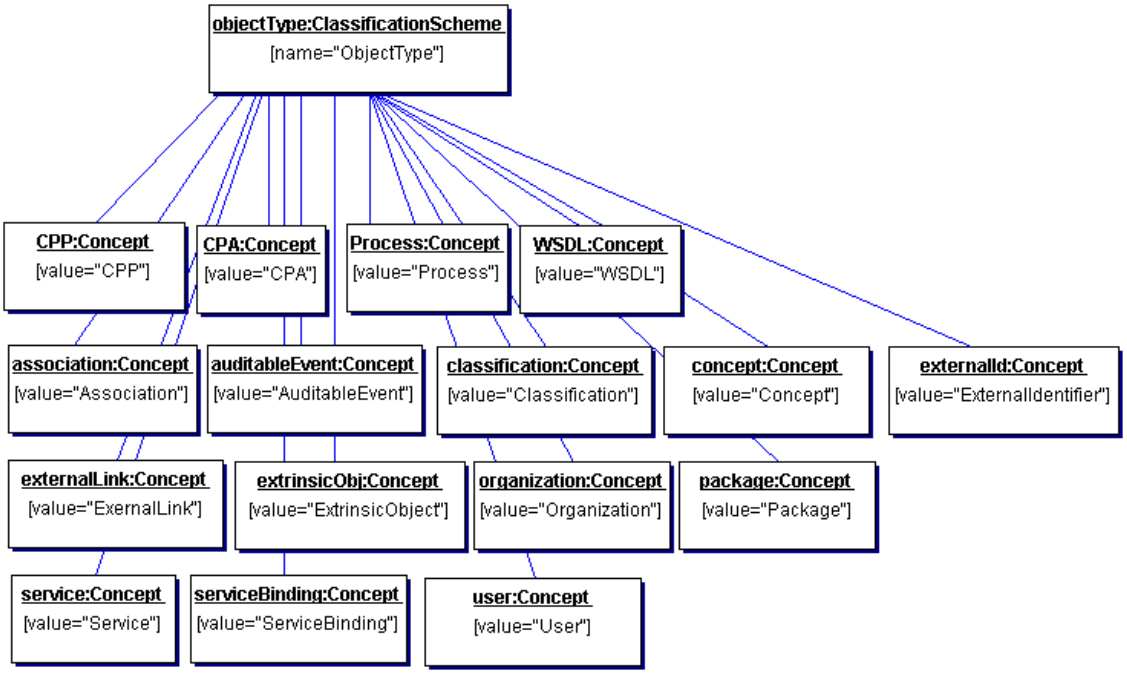


Figure 24: Pre-defined enumeration ObjectType

A.3 Enumeration PhoneType

This enumeration is used in the getPhoneType method of TelephoneNumber.

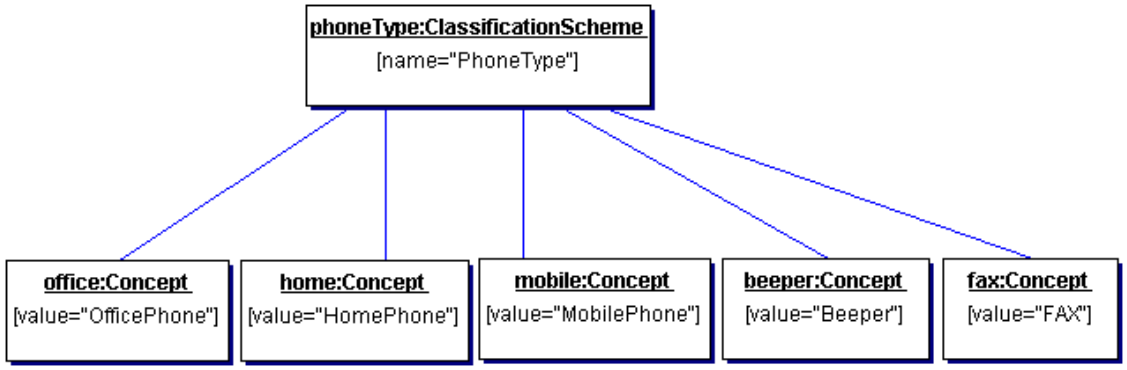


Figure 25: Pre-defined enumeration PhoneType

A.4 Enumeration AssociationType

This enumeration is used in the getAssociationType method of Association.

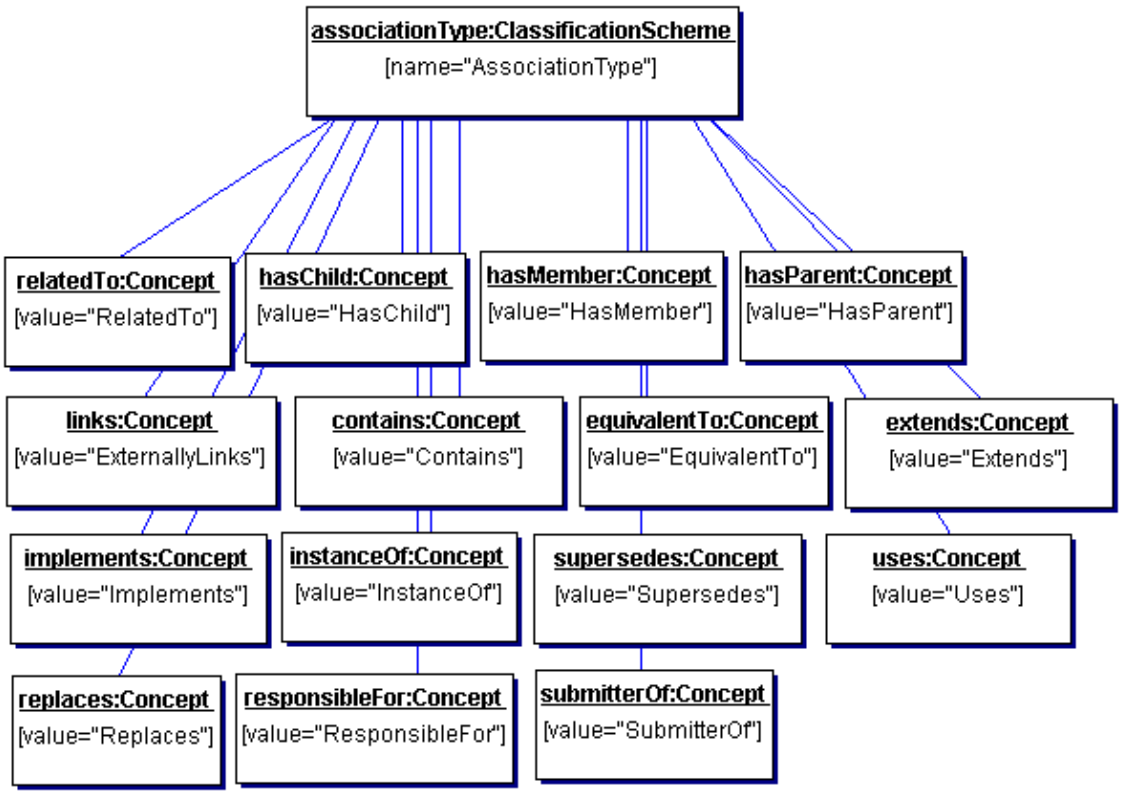


Figure 26: Pre-defined enumeration AssociationType

A.5 Enumeration URLType

This enumeration is used in classifying a ServiceBinding according to the type of access point it supports.

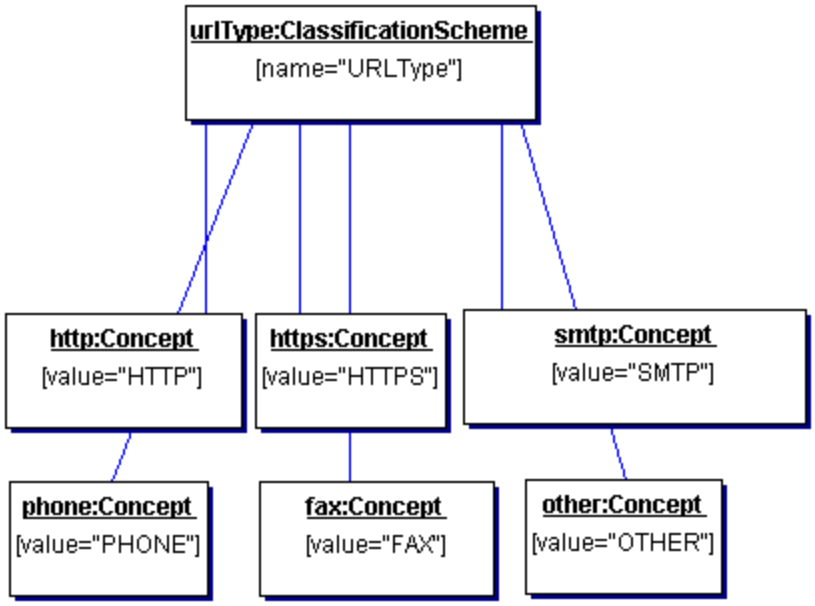


Figure 27: Pre-defined enumeration URLType

A.6 Enumeration PostalAddressAttributes

This enumeration is used to provide a mapping from the attributes of the PostalAddress class to any user-defined taxonomy.

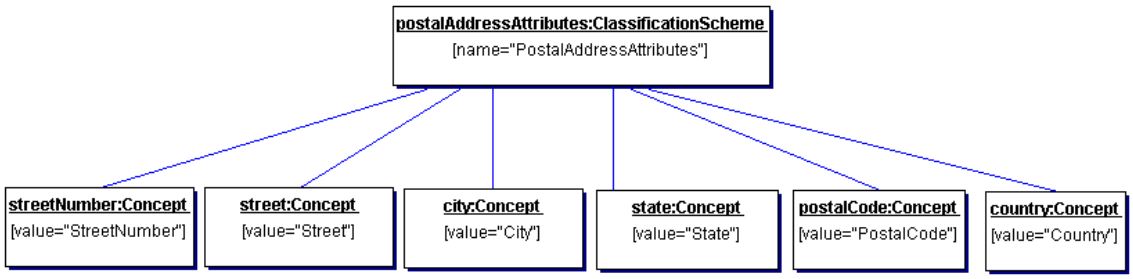


Figure 28: Pre-defined enumerations for PostalAddressAttributes

Appendix B Semantic Equivalence of JAXR Concepts

This appendix describes those aspects of the JAXR API that allow the definition of semantic equivalence between two Concepts in potentially two different ClassificationSchemes.

When two Concepts are semantically equivalent, they represent the same notion or concept. It does not matter which node is the source and which is the target, since the semantics of this association implicitly apply bi-directionally.

An example of the use of semantic equivalence is in the PostalAddress mapping for UDDI providers.

A JAXR provider must allow a client to define semantic equivalences on a per-connection basis using the `javax.xml.registry.semanticEquivalences` connection property. A JAXR provider may provide the ability to define the semantic equivalences in a provider-specific manner as global defaults for the provider.

Each semantic equivalence is specified as a separate tuple in the single `javax.xml.registry.semanticEquivalences` connection property. Each individual semantic equivalence tuple consists of the id of two equivalent concepts separated by a comma ‘,’ separator. Within the property value, individual semantic equivalence tuples are separated by the ‘|’ character.

The format is described below:

```
javax.xml.registry.semanticEquivalences=<id1>, <id2> | <id3>, <id4> | ….
```

Spaces are allowed between tokens. The backslash ‘\’ character may be used as a continuation indicator, as allowed by Java property file syntax.

An example follows below. Each property is on the same line but is shown wrapped due to the formatting limits of this document.

A JAXR provider must ignore a semanticEquivalence property that is invalid for some reason (for example, the id specified was not that of a Concept). In such cases, the JAXR provider should emit a suitable warning to the user.

```
javax.xml.registry.semanticEquivalences= \
urn:uuid:0a1324f7-6d4a-4d73-a088-9ab1d00c9a91, \
urn:uuid:23a5feac-26b9-4525-82fc-997885a0e6a2 | \
urn:uuid:1acf6ed2-cd6e-4797-aad8-8937a3cff88b, \
urn:uuid:152d6f28-cb56-4e5d-9f55-96b132def0e4
```

Appendix C JAXR Mapping to ebXML Registry

This appendix describes how the JAXR information model maps to the ebXML Registry information model defined in [ebXML-RIM]. Note that the JAXR information model used [ebXML-RIM] as its starting point. Consequently, the mapping from JAXR to ebXML Registry is often direct.

C.1.1 Mapping of Interfaces

Table 1 describes the mapping between the interfaces in the ebXML and JAXR information models respectively. Both models use UML interfaces and classes.

Table 1: Mapping of ebXML Interfaces to JAXR Interfaces

EbXML	JAXR	Description
Association	Association	Identical definition
AuditableEvent	AuditableEvent	Identical definition
Classification	Classification	Identical definition
ClassificationNode	Concept	Name change only
EmailAddress	EmailAddress	Identical definition
ExternalIdentifier	ExternalIdentifier	Identical definition
ExternalLink	ExternalLink	Identical definition
ExtrinsicObject	ExtrinsicObject	Identical definition
Organization	Organization	Identical definition
RegistryPackage	RegistryPackage	Identical definition
PersonName	PersonName	Identical definition
PostalAddress	PostalAddress	Identical definition
RegistryEntry	RegistryEntry	Factored slots-related methods into ExtensibleObject
RegistryObject	RegistryObject	Change name of Id attribute to Key
Service	Service	Identical definition
ServiceBinding	ServiceBinding	Identical definition
Slot	Slot	Identical definition

SpecificationLink	SpecificationLink	Identical definition
TelephoneNumber	TelephoneNumber	Identical definition
User	User	Identical definition
Versionable	Versionable	Identical definition

C.1.2 Mapping of New Classes In JAXR To ebXML

JAXR	Description
ExtensibleObject	Factored slots-related methods from RegistryObject into ExtensibleObject. No impact on mapping.
Key	Maps to an id of type String. No real impact on mapping.

C.1.3 ebXML Functionality Not Supported By JAXR

The following table declares all ebXML functionality that is not accessible via JAXR API. Any potential omissions from this list are specification errors and should be reported.

ebXML Feature	Disposition	Description
		Currently, all functionality of OASIS ebXML Registry is supported.

Appendix D JAXR Mapping To UDDI

This appendix describes how the JAXR information model maps to the UDDI XML data structure as defined in version 2.0 of the UDDI specification [UDDI-DS]. UDDI data structures are described in an XML format.

D.1 Mapping of UDDI Inquiry API Calls To JAXR

The following table shows the mapping from UDDI Inquiry API methods to JAXR methods. Unless otherwise qualified, the JAXR interface is BusinessQueryManager.

UDDI Method	Business-QueryManager Method	Comments
find_binding	findServiceBindings	No comments
find_business	findOrganizations	No comments
find_related_business	findAssociations	Will require traversing the association to get the related business in separate API call
find_service	findServices	No comments
find_tModel	findConcepts, findClassification-Schemes	No comments
get_bindingDetail	Not needed	Handled transparently by JAXR provider
get_businessDetail	Not needed	Handled transparently by JAXR provider
get_businessDetailExt	Unsupported	Use RegistryService. makeRegistrySpecificRequest
get _serviceDetail	Not needed	Handled transparently by JAXR provider
get_tModelDetail	Not needed	Handled transparently by JAXR provider

D.2 Mapping of UDDI Publisher API Calls to JAXR

The following table shows the mapping from UDDI Publisher API methods to JAXR methods. Unless otherwise qualified, the JAXR interface is BusinessLifeCycleManager.

UDDI Method	BusinessLifeCycle-Manager Method	Comments
add_publisherAssertions	saveAssociations, confirmAssociation	No comments
delete_binding	deleteServiceBindings	No comments

delete_business	deleteOrganizations	No comments
delete_publisherAssertions	deleteAssociations	No comments
delete_service	deleteServices	No comments
delete_tModel	deleteClassification-Schemes and deleteConcepts	In UDDI delete_tModel does not delete the tModel. It simply hides it from find_tModel calls. The QueryManager.getRegistry-Object calls will still return the deleted tModel after a deleteConcepts or deleteClassificationSchemes call.
discard_authToken	Not needed	Handled transparently by JAXR provider
get_assertionStatusReport	BusinessQuery-Manager.-findCallerAssociations	No comments
get_authToken	Not needed	Handled transparently by JAXR provider
get_publisherAssertions	BusinessQuery-Manager.-findCallerAssociations	JAXR provider must transparently authenticate with UDDI provider
get_registeredInfo	QueryManager. getRegistryObjects(objectType)	JAXR provider must transparently authenticate with UDDI provider
save_binding	saveServiceBindings	No comments
save_business	saveOrganizations	No comments
save_service	SaveServices	No comments
save_tModel	saveClassificationSchemes and saveConcepts	No comments
set_publisherAssertions	saveAssociations	No comments

2162

D.3 Simplified UML Model For UDDI Information Model

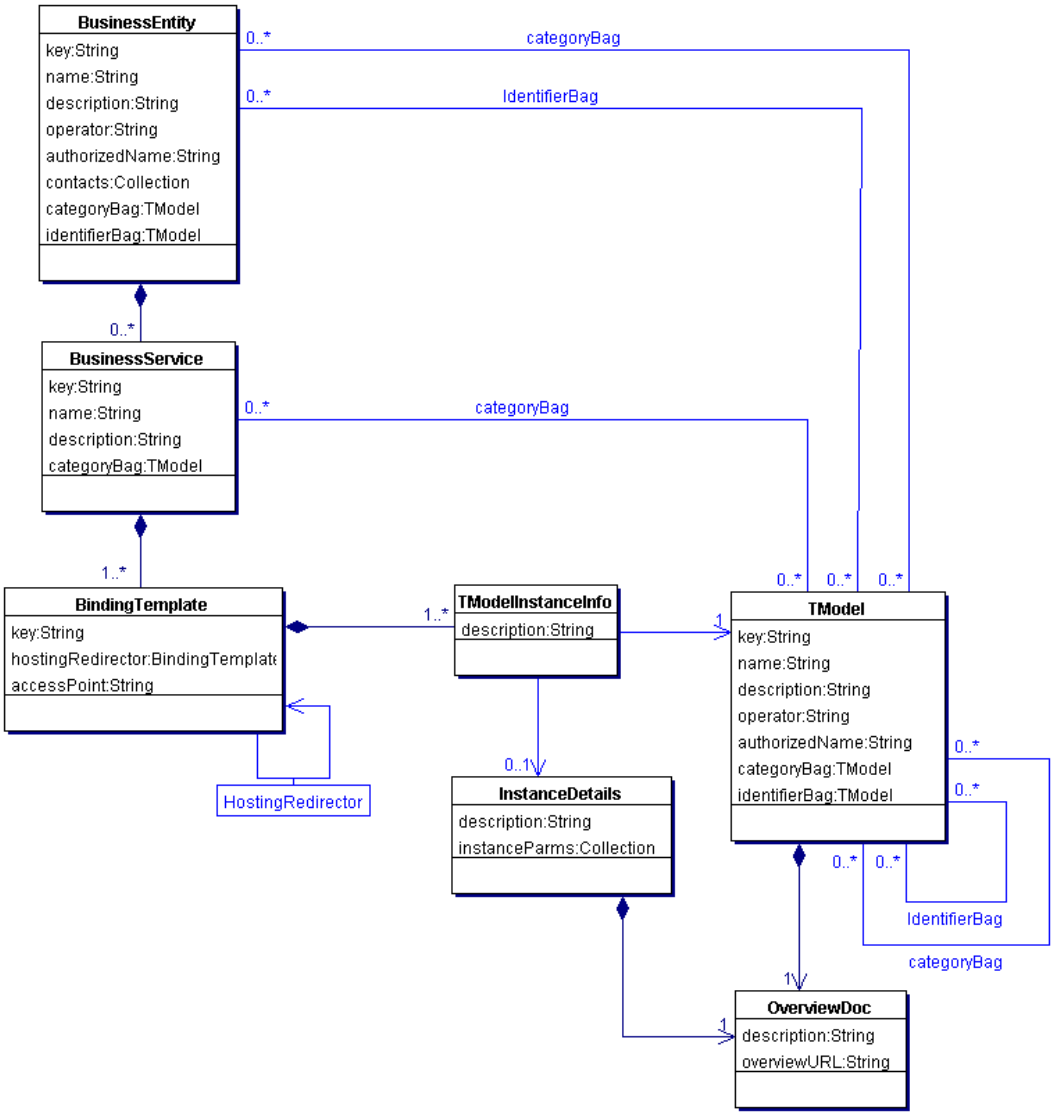


Figure 29: Simplified UML Model for UDDI Information Model

Figure 29 shows a simplified UML model representing the UDDI information model. Note that the model is not an exact rendering of the UDDI information model but has been simplified to aid the reader's understanding.

D.4 Mapping of JAXR Attributes to UDDI

The UDDI data types are not extensible. It is therefore not always possible to map an attribute from a JAXR interface to UDDI. For example, UDDI does not support majorVersion and minorVersion. A JAXR provider for UDDI must throw an UnsupportedCapabilityException when a client attempts to call a setter method for an attribute that has no mapping in UDDI (e.g. RegistryEntry.setMajorVersion). Note that such methods are designated as level 1 methods for the convenience of the JAXR client programmer.

Similarly, not all JAXR interfaces have a mapping to UDDI. For example, the JAXR RegistryPackage interface has no mapping to UDDI. The LifeCycleManager.createObject method and any other related factory methods must throw an UnsupportedCapabilityException when a client attempts to create an object that cannot be mapped to UDDI.

D.5 Mapping of UDDI Attributes to JAXR

The JAXR specification used the following approaches, listed in order of preferences, when specifying the mapping between UML attributes of the JAXR model and UDDI:

1. Map UDDI attribute to a statically defined (non-Slot) JAXR attribute within a JAXR interface
2. Map UDDI attribute to a dynamically defined Slot attribute within an instance of a JAXR interface

D.6 Mapping of Interfaces

This section provides the mapping of the highest-level UDDI data structures to the interfaces defined by the JAXR information model. The table provides an entity-level mapping and subsections discuss element/attribute-level mapping for each key concept. Since JAXR defines its information model in terms of interfaces, most of the UDDI entity attributes are mapped to operations on the JAXR information model objects.

UDDI	JAXR	Description
businessEntity	Organization	
businessService	Service	
bindingTemplate	ServiceBinding	See D.6.3 for details.

tModel (fingerprint)	Concept	See D.6.4 for details.
tModel (namespace)	ClassificationScheme	See D.6.4 for details.
discoveryURL	ExternalLink	See D.6.1.1 for details.
contact	User	
identifierBag	Collection of ExternalIdentifier instances	See D.7.2 for details.
categoryBag	Collection of Classification instances	See D.7.3 for details.
address	PostalAddress	See D.6.1.3 for details.
overviewDoc	ExternalLink	See D.6.4.5 for details.
keyedReference (in categoryBag)	Classification	See D.7.1 for details.
keyedReference (in identityBag)	ExternalIdentifier	See D.7.1 for details.

2199

2200 ### D.6.1 UDDI businessEntity

2201 businessEntity is one of the four core data structures in UDDI. The
2202 businessEntity maps to Organization in the JAXR information model. The
2203 following table shows the attribute level mapping between a UDDI businesEntity
2204 and a JAXR Organization.
2205

businessEntity	Organization	Description
businessKey	Organization.getKey	
authorizedName	Organization.getSlot	Read-only Slot named *authorizedName* of type String.
operator	Organization.getSlot	Read-only Slot named *operator* of type String.
discoveryURL	Organization.getExternalLinks	businessEntity contains a list of discoveryURLs while Organization contains a collection of external links.

name	Organization.getName	
description	Organization.getDescription	
contact	Organization.getUsers	JAXR designates one contact as the primary contact while UDDI does not. The mapping will assume the first UDDI contact to be the primary contact in JAXR.
businessServices	Organization.getServices	businessService maps to Service interface.
identifierBag	Organization. getExternalIdentifiers	See D.7.2.
categoryBag	Organization. getClassifications	See D.7.3.

D.6.1.1 UDDI discoveryURL

In UDDI, the default discoveryURL is assigned by the UDDI registry and is used to retrieve the XML document for the businessEntity and everything contained within it. Any additional discoveryURLs are assigned by the submitter, and provide links to external content that provides information about the businessEntity (referred to as discovery documents in UDDI).

The default discoveryURL is identified by having a useType of either businessEntity or businessEntityExt. Any other useType value indicates an optional discoverURL.

D.6.1.1.1 Getting a discoveryURL from UDDI

When a businessEntity is retrieved from UDDI and mapped to a JAXR Organization, all discovery URLs are mapped to an ExternalLink. The first ExternalLink in the Collection returned by getExternalLinks method on Organization object must map to the default registry provider assigned discoveryURL. All other ExternalLinks must map to the optional discoveryURLs.

D.6.1.1.2 Saving discoveryURL to UDDI

When a client saves an Organization to UDDI, the default discoveryURL is assigned to the corresponding businessEntity by the registry provider and is not provided by the JAXR client. All ExternalLink instances associated with the Organization are mapped to optional discoveryURL instances such that the name of the ExternalLink is mapped to the useType attribute of the discoveryURL.

DiscoveryURL	ExternalLink	Description
useType	ExternalLink.getName	
url value	ExternalLink.getExternalURI	

D.6.1.2 UDDI contact

The UDDI contact element maps to the interface User in JAXR as follows:

Contact	User	Description
useType	User.getType	
description	User.getDescription	
personName	User.getPersonName.getFullName	
phone	User.getTelephoneNumbers.getNumber	
email	User.getEmailAddresses	
address	User.getPostalAddress	

D.6.1.3 UDDI address

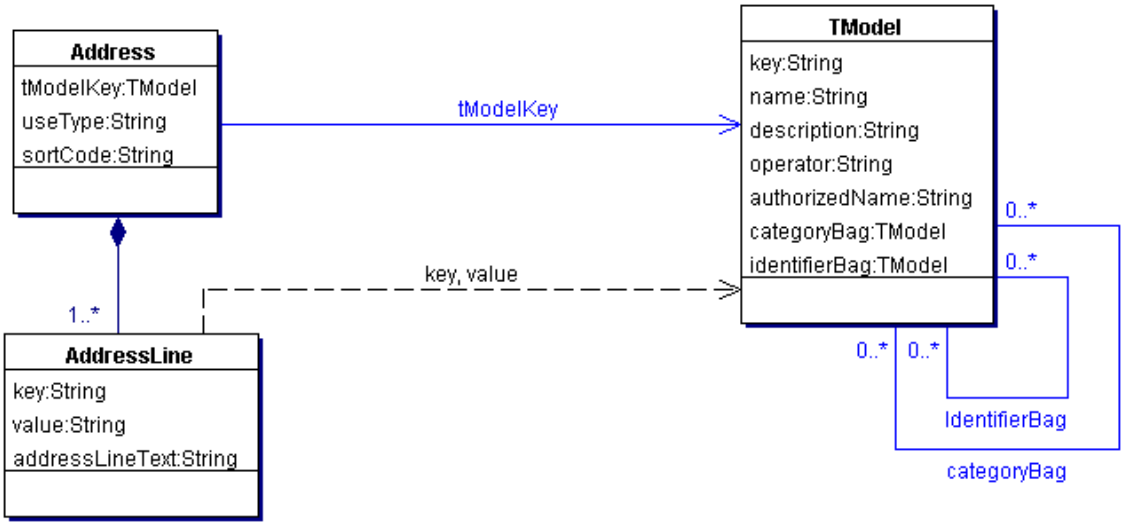


Figure 30: UDDI Information Model for Address

A UDDI address has an ordered Collection of addressLine instances. Each addressLine is a String. However in UDDI V2, each addressLine may be attached a meaning by linking it via a key-value pair of attributes to a taxonomy element defined under a taxonomy represented by a tModel. The tModel is referenced by a tModelKey within the address. Thus in UDDI an address is unstructured by default and can optionally be given meaning.

In JAXR, PostalAddress is a structured interface with well-defined attributes for street, city, postal code, country etc. This brings about an issue of mapping between structured information in JAXR and unstructured information in UDDI. The solution is to use semantic equivalence mapping capabilities in JAXR API as described in Appendix B, as follows.

1. The user or system administrator defines one or more user-defined ClassificationSchemes (a.k.a postal address schemes) representing UDDI tModels commonly used by address as a postal address scheme.
2. The user or system administrator defines semantic equivalence between each Concept in the pre-defined PostalAddressAttributes ClassificationScheme and one or more Concepts in the user-defined postal scheme used by address.

3. The client sets a default postalAddressScheme (using the javax.xml.registry.postalAddressScheme connection property) on the Connection instance and/or sets a postal address scheme on a specific PostalAddress. The postal address scheme for a PostalAddress overrides the default postal scheme set on the Connection.

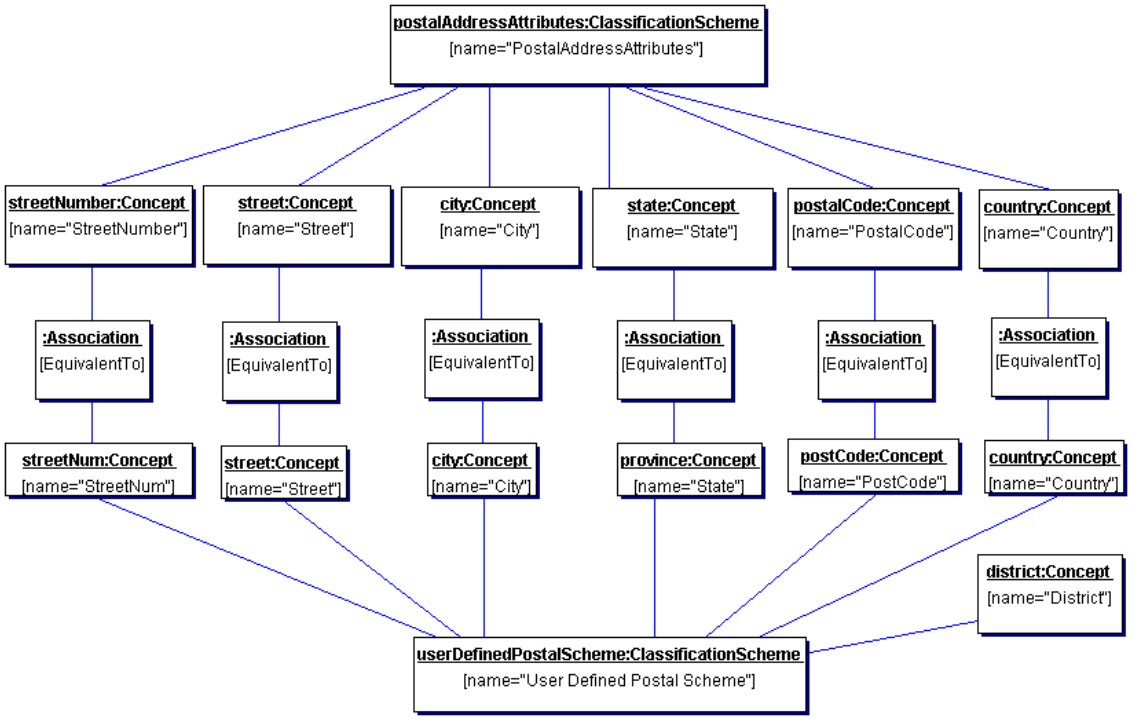


Figure 31: Semantic Equivalence and Mapping of User Defined Postal Scheme to PostalAddress Attribute

A JAXR provider for UDDI must use the semantic equivalences defined with user-defined ClassificationScheme to validate and properly map PostalAddress attributes to addressLines with keyed references to the tModel corresponding to the postal scheme for the PostalAddress. This allows a JAXR client programmer to use the set/get methods defined in the PostalAddress class to conveniently set and get the values of the PostalAddress attributes such as city, stateOrProvince, country etc.

D.6.1.3.1 Mapping of PostalAddress During Save Operations

The JAXR UDDI provider must map the PostalAddress to UDDI during a Save Operation as follows:

- The PostalAddress must map to an address element.

- If there is a postal scheme defined on the PostalAddress, or if a default postal scheme is defined on the RegistryService, then a tModelKey attribute must be set to the Id of the postal scheme. Otherwise, the tModelKey must not be specified.
- Each attribute of PostalAddress must map to an addressLine element.
- The order of addressLine elements must match the order of PostalAddressAttributes as shown left to right in Figure 31 above for the Concepts under postalAddressAttributes scheme (second row from top).
- If a tModelKey has been set on the address element and there is a semantic equivalence found between a PostalAddress attribute and the postal scheme, then a name and value attribute pair is specified for the corresponding addressLine element. The name and value are the name and value of the Concept in the postal scheme that was equivalent to the PostalAddress attribute. If no semantic equivalence was found then the name attribute and the value attribute must be as the name and value of the corresponding Concept in PostalAddressScheme.
- If no tModelKey has been set on the address element then each addressLine element should specify the name and value attribute as defined by the name and value of the corresponding Concept in PostalAddressScheme.

D.6.1.3.2 Mapping of UDDI address During Find Operations

The JAXR UDDI provider must map the UDDI address element to JAXR during a find operation as follows:

- The address element must map to a PostalAddress instance.
- If there is a tModelKey defined for the address element and it matches a postal scheme, the postalScheme must be set to the matching postal scheme.
- An address line is mapped to a PostalAddress attribute if a match to a PostalAddress attribute is found. The match is found if the following conditions are true:
  - A postal scheme has been set and
  - The value attribute of the addressLine matches a value attribute of a Concept in the postal scheme and
  - That postal scheme concept with matching value has a semantic equivalence with a PostalAddress attribute

2307 o An address line is mapped to a value in the Collection of values in a slot
2308 named addressLines defined for the PostalAddress if no match is found to
2309 a PostalAddress attribute.

2310

2311 The following table summarizes the attribute mappings between the UDDI
2312 address element and JAXR PostalAddress interface.

2313

Address	PostalAddress	Description
addressLines	PostalAddress.getSlot	Mapped to attributes of PostalAddress if semantic equivalence established. Otherwise mapped to a Slot named addressLines which has a Collection of values
useType	PostalAddress.getType	
sortCode	PostalAddress.getSlot	Slot named sortCode

2314

2315 D.6.2 UDDI businessService

2316 businessService in UDDI represents a logical group of services, which have
2317 common classifications. It is functionally minimal and really serves as a grouping
2318 of bindingTemplates.

2319 businessService maps directly to the Service interface in the JAXR information
2320 model.

2321

businessService	Service	Description
businessKey	Service.getProvidingOrganization.getKey	
serviceKey	Service.getKey	
name	Service.getName	
description	Service.getDescription	
bindingTemplates	Service.getServiceBindings	
categoryBag	Service.getClassifications	See D.7.3.

2322

2323 D.6.3 UDDI bindingTemplate

2324 The bindingTemplate is another one of the core concepts in UDDI and serves not
2325 only to provide means of accessing (an entry point to) the desired service but
2326 also to carry with it the technical specification for the service. These technical
2327 specifications may be either in the form of a text-based document or in an
2328 interface definition language such as WSDL.

2329 The UDDI bindingTemplate element maps to the JAXR ServiceBinding interface
2330 as follows:

2331

bindingTemplate	ServiceBinding	Description
bindingKey	ServiceBinding.getKey	
serviceKey	ServiceBinding.getService.getKey	
description	ServiceBinding.getDescription	
accessPoint	ServiceBinding.getAccessURI	URLType attribute in accessPoint is mapped by classifying the ServiceBinding with a sub-concept of URLType Concept (A.5). Default urlType is http.
hostingRedirector	ServiceBinding. getTargetBinding	There is only one element, bindingKey, in this structure and it maps to targetBinding attribute
tModelInstanceDetails	Not mapped explicitly as it is just a Collection	
tModelInstanceInfo	Mapped to a SpecificationLink	See D.10 for mapping example.
instanceDetail	Mapped to a SpecificationLink	See D.10 for

		mapping example.

2332

2333 D.6.3.1 tModelInstanceInfo and instanceDetails

2334 Both tModelInstanceInfo and instanceDetails are combined together and mapped
2335 to a single SpecificationLink instance.

2336 The registryObject attribute in the SpecificationLink instance is the Concept
2337 representing the tModel providing the specification fingerprint for the
2338 bindingTemplate.

2339 The description of the instanceDetails maps to the usageDescription attribute of
2340 the SpecificationLink instance. The instanceParms maps to the
2341 usageParameters attributes of the SpecificationLink instance. Note that JAXR
2342 allows multiple usageParameters while UDDI allows a single instanceParms.
2343 Thus a UDDI provider must throw an InvalidRequestException if a client attempts
2344 to set more than one usageParameter on a SpecificationLink instance.

2345 D.6.4 tModel

2346 In UDDI, tModel is an overloaded concept that can be used for a few different
2347 purposes. The following are two broad categories of purposes that tModels serve
2348 in UDDI:

2349 o To serve as a namespace for a taxonomy (e.g. NAICS) or identification
2350 scheme (DUNS)

2351 o To serve as a fingerprint or proxy for a technical specification that lives
2352 outside the registry in a bindingTemplate

2353 In the JAXR the above two uses of tModel are modeled separately. The
2354 namespace use is modeled with the ClassificationScheme interface, while the
2355 technical fingerprint use is modeled with any RegistryObject which in case of a
2356 UDDI provider must be a Concept. The
2357 `SpecificationLink.getSpecificationObject` method must return a
2358 Concept instance for a UDDI provider.

2359 D.6.4.1 tModel Mapping to ClassificationScheme

tModel	Concept	Description
tModelKey	ClassificationScheme.getKey	
authorizedName	ClassificationScheme.getSlot	Read-only Slot named

		authorizedName
operator	ClassificationScheme.getSlot	Read-only Slot named *operator*
name	ClassificationScheme.getName	
description	ClassificationScheme.getDescription	
overviewDoc	ClassificationScheme.getExternalLinks	See D.6.4.4.
identifierBag	ClassificationScheme.getExternalIdentifiers	See D.7.2.
categoryBag	ClassificationScheme.getClassifications	See D.7.3.

2360

2361 D.6.4.2 tModel Mapping to Concept

2362

tModel	Concept	Description
tModelKey	Concept.getKey	
authorizedName	Concept.getSlot	Read-only Slot named *authorizedName*
operator	Concept.getSlot	Read-only Slot named *operator*
name	Concept.getName	
description	Concept.getDescription	
overviewDoc	Concept.getExternalLinks	See D.6.4.4.
identifierBag	Concept.getExternalIdentifiers	See D.7.2.
categoryBag	Concept.getClassifications	See D.7.3.

2363

2364 D.6.4.3 Mapping of tModels During JAXR Find Operations

2365 During JAXR find operations, the JAXR provider must be able to determine
2366 whether a tModel in UDDI is a namespace tModel or whether it is a fingerprint
2367 tModel. This is necessary in order to decide whether to map the tModel to a
2368 ClassificationScheme or Concept.

The JAXR UDDI provider must use the catgegoryBag information association with a UDDI tModel in order to make the correct mapping to a JAXR ClassificationScheme or Concept.

A JAXR provider must map a UDDI tModel to a ClassificationScheme if the tModel's categoryBag has a keyedReference that uses the well-known `uddi-org:types` taxonomy in UDDI (with tModelKey uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4) for its tModel and uses any of the following taxonomy values:

- Identifier
- Namespace
- Categorization
- PostalAddress

In all other cases, a JAXR provider must map a UDDI tModel to a Concept.

Note that it is possible that a UDDI tModel was intended to be a ClassificationScheme but was not properly categorized in UDDI. In such cases the tModel would be mapped to a Concept instead of a ClassificationScheme. The user must explicitly determine which such Concepts are actually ClassificationSchemes and then use the createClassificationScheme(Concept) method of LifeCycleManager to safely cast the Concept to a ClassificationScheme. Note that such cases indicate problems within UDDI content and should be reported to the content's owner.

D.6.4.4 Mapping to tModels During JAXR Save Operations

During JAXR Save operations, ClassificationSchemes and Concepts that have no parent or ClassificationScheme are mapped to tModels.

It is suggested but not required that a JAXR provider for UDDI attempt to categorize tModels based upon information available on their intended usage. ClassificationSchemes related tModels *may* be automatically categorized by the well-known `uddi-org:types` taxonomy in UDDI (with tModelKey uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4) as follows:

- PostalAddress schemes are assigned the taxonomy value of PostalAddress
- ClassificationSchemes used for classification purposes are assigned the taxonomy value of categorization
- ClassificationSchemes used for identification purposes are assigned the taxonomy value of identification

- If there is not enough information then the default taxonomy value is categorization.

Specification related tModels mapped from Concept may be automatically categorized by the well-known `uddi-org:types` taxonomy in UDDI (with tModelKey uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4) as follows:

- The keyed reference is assigned a taxonomy value of *specification*.

D.6.4.5 overviewDoc

An overviewDoc maps to an ExternalLink in JAXR. The ExternalLink is associated with the Concept or ClassificationScheme that the tModel is mapped to. Since UDDI allows only one overviewDoc on a tModel or a instanceDetails, a JAXR UDDI provider must throw UnsupportedCapabilityException if more than one ExternalLink is added to a Concept (tModel), ClassificationScheme (tModel) or SpecificationLink (instanceDetails). A JAXR UDDI provider should also throw UnsupportedCapabilityException if an ExternalLink is added to any other object besides Organization, Concept, ClassificationScheme or SpecificationLink.

OverviewDoc	ExternalLink	Description
description	ExternalLink.getDescription	
overviewURL	ExternalLink.getExternalURI	

D.7 Mapping of Common Data Types

So far, we have described the highest-level mapping between the main data structures in UDDI and interfaces in the JAXR information model. The subsequent section describes the mapping between data structures that are commonly reused in UDDI and the JAXR API.

D.7.1 keyedReference

A keyedReference element is used either to contain a group of classifications or to contain a group of identifiers for an object. To that end, keyedReference can map to either ExternalIdentifiers or Classifications. For this reason, there are actually 2 tables specifying each individual mapping.

When keyedReference is being used in an identifierBag, it is mapped to a JAXR ExternalIdentifier. A JAXR information model object being marshaled to XML for

2435 UDDI will have all its external identifiers marshaled into identifierBag. Similarly,
2436 keyedReferences in an identifierBag in a UDDI response are un-marshaled into
2437 ExternalIdentifiers by the JAXR provider.
2438

keyedReference	ExternalIdentifier	Description
tModelKey	ExternalIdentifier.getIdentificationScheme().getKey	From the RegistryObject
keyName	ExternalIdentifier.getName	From the RegistryObject. This is symbolic (such as Tax Id).
keyValue	ExternalIdentifier.getValue	This is the unique id (e.g. tax id), which identifies the ebusiness entity.

2439
2440 When keyedReference is used in a categoryBag, it is mapped to a JAXR
2441 Classification.
2442
2443 When a JAXR object is being marshaled to XML for UDDI, all its Classifications
2444 are marshaled into categoryBag according to the mapping described below.
2445 When the Classification uses an internal taxonomy, the JAXR provider for UDDI
2446 must validate all keyValues in the resulting UDDI keyedReference.
2447
2448 In UDDI a keyedReference can have only one keyName. In contrast in the JAXR
2449 API, a Classification or an ExternalIdentifier may have multiple key names as
2450 defined by the name attribute. When a keyedReference in mapped to a JAXR
2451 Classification or an ExternalIdentifier, the keyValue must be mapped to the
2452 LocalizedString in the default Locale for the client. When a JAXR Classification or
2453 an ExternalIdentifier is mapped to a UDDI keyedReference, the keyValue is
2454 chosen using the following precedence rules:
2455

2456 1. Use the value specified in LocalizedString for the default locale if available
2457 2. Use the value specified in LocalizedString for the en_US locale if available
2458 3. Use the first available name in any locale if available
2459 4. If all of above fail then do not specify the keyName

2460

keyedReference	Concept	Description

tModelKey	Classification.getClassificationScheme.getKey	
keyName	Classification.getName	This is the name of a taxonomy element
keyValue	Concept.getValue	This is the value of identifying a taxonomy element

2461

2462 D.7.2 identifierBag

2463 identifierBag is a collection of keyedReferences. An identifierBag is modeled in
2464 the JAXR information model as a Collection of ExternalIdentifiers.

2465 D.7.3 categoryBag

2466 categoryBag is a collection of keyedReferences. A categoryBag is modeled in
2467 the JAXR information model as a Collection of Classifications.

2468 D.7.4 tModelBag

2469 tModelBag is a collection of tModel *uuid_key* values that represents the technical
2470 *fingerprint* of a bindingTemplate structure contained within the businessService
2471 specified by the serviceKey value.

2472 A tModelBag is modeled in the JAXR information model as a Collection of
2473 Concepts that represent technical fingerprint Concepts that serve as proxies for
2474 technical specification is a ServiceBinding (bindingTemplate).

2475 D.8 Mapping of UDDI phone Element

2476 UDDI allows a single String for the entire phoneNumber and an optional
2477 useType. The JAXR TelephoneNumber class provides a more structured
2478 representation of the phone number. Therefore, for JAXR UDDI providers the
2479 only relevant attributes are phoneType and number.

D.8.1 Mapping of phone During Save Operations

A JAXR UDDI provider must throw an UnsupportedCapabilityException if a client programmer tries to call any of the following operations:

- setAreaCode
- setCountryCode
- setExtension
- setURL

The client programmer is expected to mainly use the setNumber method to set the complete telephone number as unstructured free-form text.

The client programmer may also set a type on the TelephoneNumber using setType method. In this case, the type specified should be used in the useType attribute for the UDDI phone element.

D.8.2 Mapping of phone During Find Operations

A JAXR UDDI provider must throw an UnsupportedCapabilityException if a client programmer tries to call any of the following operations:

- getAreaCode
- getCountryCode
- getExtension
- getURL

A JAXR UDDI provider must map the CDATA of the phone element to the number attribute of telephoneNumber.

If a useType is present for the phone element, then a JAXR UDDI provider must map the useType to the type attribute of TelephoneNumber.

The client programmer is expected to mainly use the getNumber method to get the complete telephone number as unstructured free-form text.

If the client programmer calls the getType method they should either get a type String or they should get null.

D.9 Mapping of name to PersonName

UDDI personName element allows a single String for the entire name of a person. The JAXR PersonName class provides a more structured representation of the a person's name. Therefore, for JAXR UDDI providers the only relevant attributes is fullName.

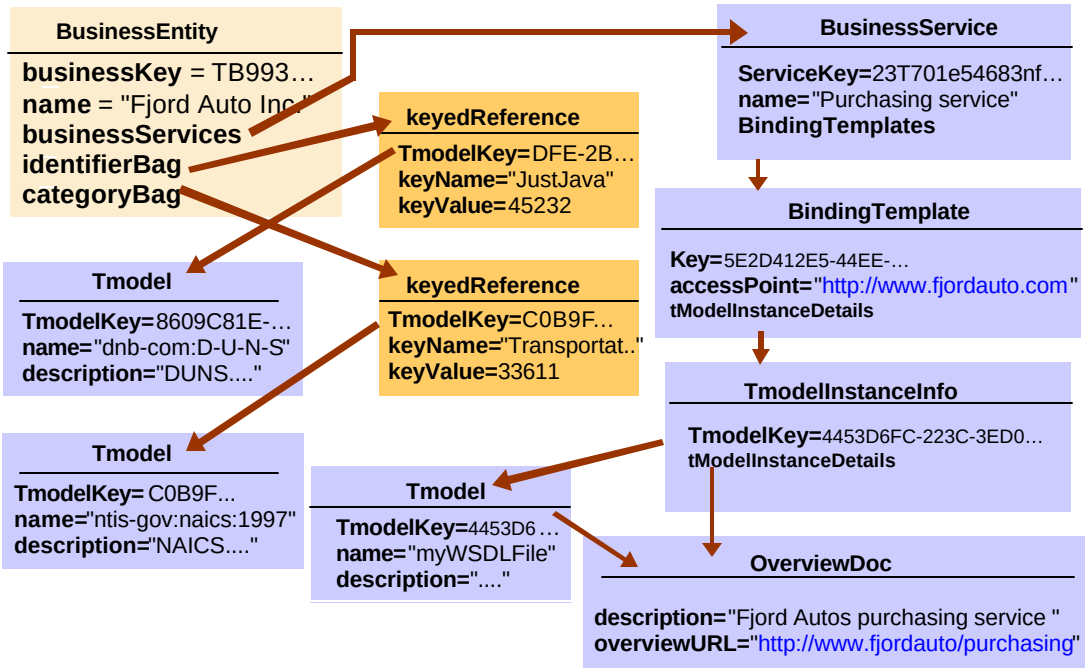
A JAXR UDDI provider must throw an UnsupportedCapabilityException if a client programmer tries to call any method of the PersonName class other than the following operations:

- getFullName
- setFullName

D.10 Example of JAXR-UDDI Mapping

Figure 32 below shows a simplified example described in terms of UDDI data structures. Figure 33 then shows the same example in terms of JAXR information model using the mapping described above.

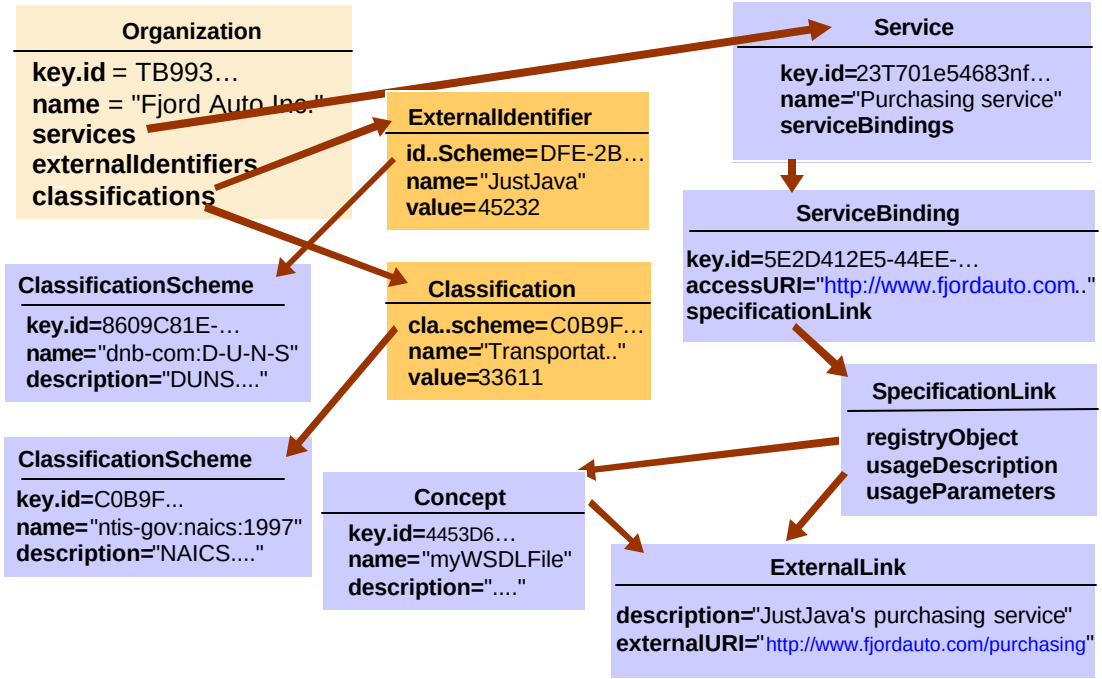
In this example a UDDI businessEntity is classified by an external classification using the taxonomy element with name "*Automobile and Light Duty Motor Vehicle Manufacturing*" and value 33611 in the NAICS taxonomy. It is also identified using a DUNS number of 45232 using the DUNS identification scheme. The businessEntity has a single businessService for a purchasing service that has a single bindingTemplate that has a single specification document that is a WSDL file.



2529

2530 Figure 32: Example in terms of UDDI Data Structures

2531



2532

2533 **Figure 33: UDDI Example Mapped to JAXR**

D.11 Provider Generated id Attributes

Due to differences between the information models of JAXR and UDDI, there are several cases where a JAXR interface must have an id while its counterpart in UDDI does not have an id defined.

In such cases, the JAXR provider must deal with this impedance mismatch transparent to the user, and generate id values in a deterministic manner.

This following table itemizes each case where id needs to be generated and suggests a nor-normative algorithm that may be used to generate ids for each such case. Note that text is wrapped due to shortage of horizontal space in columns.

JAXR Interface	Algorithm	Example
ExternalIdentifier	<RegistryObjectId>: <identificationSchemeId>: <value>	a2345678-1234-1234-123456789012:a2345678-1234-1234-123456789013:Social Security Number
Association	<sourceObjectId>: <targetObjectId>: <associationType>	a2345678-1234-1234-123456789012:a2345678-1234-1234-975123456789013:Supersedes
Classification (internal)	<classifiedObjectId>: <classificationNodeId>	a2345678-1234-1234-123456789012:a2345678-1234-1234-123456789013
Classification (external)	<classifiedObjectId>: <classificationSchemeId>: <nodeRepresentation>	a2345678-1234-1234-123456789012:a2345678-1234-1234-123456789013:61
ExternalLink	<externalURI>:<sequenceId>	http://www.sun.com:1
SpecificationLink	<serviceid>:<accessURI>: <targetBindingId>: <sequenceId>: <specificationObjectId>	a2345678-1234-1234-123456789012: http://www.sun.com::1: a2345678-1234-1234-123456789013

D.12 Supporting Taxonomy Service In JAXR UDDI Providers

D.12.1 Normative Description

This section provides a normative description of all required features when implementing a taxonomy service within a JAXR UDDI provider:

- A JAXR UDDI provider must support the NAICS, UNSPSC and ISO 3166 Geography taxonomies as internal taxonomies available within the implementation specific taxonomy service.
- A JAXR UDDI provider must provide a provider-specific way for users to configure and manage arbitrary taxonomies within the taxonomy server.

D.12.2 Non-normative Description

This section provides a non-normative description of how a JAXR UDDI provider may support a taxonomy service, thus allowing entire taxonomies complete with their taxonomy structure to be available as internal ClassificationSchemes.

A taxonomy service may be implemented in several forms. Below are some examples:

- The JAXR provider may read one or more client side configuration files upon startup that contain taxonomy information.
- The JAXR provider may have a server-side component that provides the taxonomy service function.
- The JAXR provider may allow a level 1 registry to be configured as the taxonomy server.

Regardless of the implementation choice for a taxonomy service internal to the JAXR UDDI provider, the following guidelines apply:

- Taxonomy information in the taxonomy server is updated or deleted via out-of-band means not described the JAXR specification.
- The JAXR UDDI provider never updates or deletes taxonomy information in the taxonomy server based upon a client call to a JAXR API method. Note that updates to the taxonomy server are done through out-of-band provider specific means.
- The `saveClassificationSchemes` and `deleteClassificationSchemes` calls in the JAXR API only affect the UDDI registry and not the taxonomy server.

- The JAXR UDDI provider must query both the taxonomy server and the UDDI registry during `findClassificationSchemes` operations. It should combine the results of querying the taxonomy server and the UDDI registry and present a unified result to the JAXR client.
- When combining results of the `findClassificationSchemes` operation, the provider must cull duplicates. In case of duplicates, the taxonomy server version should be kept since it has taxonomy structure.

D.13 UDDI Functionality Not Supported By JAXR

The following table declares all UDDI functionality that is not accessible via JAXR API. Any potential omissions from this list are specification errors and should be reported.

UDDI Feature	Disposition	Description
BusinessEntityExt functionality	This functionality is not suitable for abstraction in JAXR. No plans to provide this in JAXR.	Deliberately not supported. Use makeRegistrySpecificRequest backdoor method in RegistryService interface.

Appendix E Value-Added Features of the JAXR API

This section described some features that are available to JAXR clients that provide unique value beyond the capabilities provided by underlying registries.

E.1 Taxonomy Browsing

The JAXR API allows a JAXR client of a UDDI registry to be able to browse the full structure of taxonomies or classification schemes. This is a unique capability, which is not available to typical (non-JAXR) UDDI clients. Even though the UDDI registry does not provide clients the ability to browse taxonomy structure, a JAXR provider for UDDI enables this useful feature. A JAXR provider comes pre-configured with standard taxonomies such as NAICS, UNSPSC and ISO 3166 Geography. It can also be extended to include any other user-defined taxonomy.

E.2 Taxonomy Validation

The JAXR API automatically validates all taxonomy values when a JAXR client creates an internal Classification using an internal taxonomy. This prevents the JAXR client from creating invalid classifications.

E.3 Smart Queries

The JAXR API enables smart queries that take advantage of the knowledge of taxonomy structures within internal classifications. This enables a JAXR client to search for an Organization classified by Asia and all sub-Concepts of Asia. This enables clients to find Organizations that are directly or indirectly classified by the Asia Concept. Client may use the `getDescendantConcepts` method of the Concept interface to get all the descendents of a Concept and use them in the findOrganization query.

E.4 Enhanced Data Integrity and Validation

The JAXR API validates all URL links submitted as part of ExternalLink objects at the time of submission. The JAXR provider must ping the URL and throw an Exception if the URL is not valid and accessible. Joint research by SalCentral and WebServicesArchitect (see http://www.webservicesarchitect.com/content/articles/clark04.asp) showed that nearly 48% of all URL links in UDDI are invalid.

By validating all URLs at the time of submission, the JAXR API enhances data integrity for data submitted to UDDI and other registries.

E.5 Automatic Categorization of UDDI tModels

In UDDI, a tModel is an overloaded concept with several different usages. For this reason the UDDI specifications suggest that all tModels be categorized according their type of usage. The JAXR information model has a separate interface for each of the unique uses of a UDDI tModel. A JAXR provder for UDDI may automatically categorize these tModels in many cases.

E.6 Simplified Programming Model

There are several areas where JAXR provides a simpler programming model than compared to raw interface defined by the underlying registry. Some examples follow:

E.6.1.1 Unification of find and get Methods

[UDDI-API2] defines two sets of methods. One is a set of *find* methods and the other is a set of *get* methods. The *find* methods perform a query for Objects in UDDI and return their identifier. The client must then use *get* methods to separately retrieve the details of specific objects. The JAXR API is simpler and only provides the *find* methods. UDDI *get* methods are called transparently within the provider if the JAXR client attempts to access detailed information about an object. This lazy fetching of objects from UDDI enables the JAXR API to present a simplified programming model to the JAXR client programmer.

E.6.1.2 Generic Handling of Object

Using the Object-Oriented principle of polymorphism, the JAXR API provides several methods that allow object operations without knowing the type of the object. For example one can call `QueryManager.saveObjects` instead of more specific `save` methods in the `BusinessQueryManager` and in a single operation save many different types of objects.

E.7 Simplified User Authentication

The JAXR API allows the user to set their Credentials on the JAXR Connection and from that point on user authentication with the target registry is completely hidden from the user. In fact the JAXR API frees the client programmer from knowing which API calls requires authentication with the target registry. The JAXR provider is smart enough to know on its own that it must authenticate with a UDDI registry for *save* and *delete* operations but not for *find* and *get* operations.

E.8 Enforce No New References to Deprecated Objects

The JAXR API makes sure that no new references are allowed to be created to deprecated objects by JAXR client. This is a value-added feature for level 1 registries such as the ebXML Registry.

Appendix F Frequently Asked Questions

Question: Why do we need a new JAXR API when we have the JNDI API?

Answer: The JNDI API was designed with a very different set of requirements than JAXR. Both are abstraction APIs over existing specifications. However, the abstraction in directory services differ quite a bit from those of XML Registries used for publishing and discovery of web services. JAXR needs richer metadata capabilities for classification and association, as well as richer query capabilities.

Question: Would not be better to have enhanced the JNDI API with the added functionality of JAXR?

Answer: That option was considered. Meeting the additional requirements of XML Registries requires an elaborate information model. The JNDI API already has an existing information model that is constrained by design to address the requirements for directory services. Extending the JNDI API would overly constrain JAXR and would create backward compatibility issues for the JNDI API.

Question: Why is JAXR an abstraction API and not targeted to a specific registry such as UDDI or ebXML?

Answer: An abstraction based JAXR API provides developers the ability to write registry client programs that portable across different target registries. This is consistent with the Java philosophy of “Write Once Run Anywhere (WORA)”. It also enables value-added capabilities as described in Appendix E. These capabilities are above-and-beyond the capabilities of underlying registries. For example, a non-JAXR UDDI client does not have the ability to do taxonomy browsing, and taxonomy aware smart queries, which are available to a JAXR client for UDDI.

Question: Why does the JAXR API not use UDDI terms and concepts?

Answer: The JAXR API is not specific to UDDI or any other registry specification. It is an abstraction API that covers multiple specifications. It is designed to enable developer choice in use of a web service registry and/or repository.

The JAXR API uses UDDI terms and concepts when they fit the JAXR information model (e.g. Service, ServiceBinding, and method names in BusinessQueryManager and BusinessLifeCycleManager)

Question: Why did the JAXR information model use the ebXML Registry Information Model as its basis rather than the UDDI data structures?

Answer: The JAXR API is designed to support multiple registries. The ebXML Registry Information Model is more generic and extensible than the UDDI data structures. Because of this characteristic, it was possible to extend the ebXML Registry Information Model to satisfy the needs of UDDI and other registries.

Question: Why was the JAXR information model not designed from the ground up?

Answer: Information models take time to develop. It was easier to study an existing information model and improve upon it.

11 References

[JAXRF] JAXR developer forum: http://groups.yahoo.com/group/jaxr-discussion

[ISO] ISO 11179 Information Model

http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba7852566210054419d7/b83fc7816a6064c68525690e0065f913?OpenDocument

[ebXML-RSS] ebXML Registry Services Specification

http://www.ebxml.org/project_teams/registry/private/registryServicesSpecificationv1.0.pdf

[ebXML-RIM] ebXML Registry Information Model

http://www.ebxml.org/project_teams/registry/private/registryInfoModelv1.0.pdf

[UDDI-DS] UDDI Version 1.0 Data Structure Specification

http://www.uddi.org/pubs/DataStructure-V1.pdf

[UDDI-DS2] UDDI Version 2.0 Data Structure Reference

http://www.uddi.org/pubs/DataStructure-V2.00-Open-20010608.pdf

[UDDI-API] UDDI Version 1.0 Programmers API

http://www.uddi.org/pubs/ProgrammersAPI-V1-1.pdf

[UDDI-API2] UDDI Version 2.0 API Specification

http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf

[SQL] Structured Query Language (FIPS PUB 127-2)

http://www.itl.nist.gov/fipspubs/fip127-2.htm

[SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules (SQL/PSM) [ISO/IEC 9075-4:1996]

IANA] IANA (Internet Assigned Numbers Authority).
Official Names for Character Sets, ed. Keld Simonsen et al.
ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

[REC-XML] W3C Recommendation. Extensible Markup language (XML)1.0(Second Edition)
http://www.w3.org/TR/REC-xml

[UUID] DCE 128 bit Universal Unique Identifier
http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20

http://www.opengroup.org/publications/catalog/c706.htm

[Futures1] Mohr, E.; Kranz, D; Halstead, R.: Lazy Task Creation: A Technique

for Increasing the Granularity of Parallel Programs. IEEE Transactions on Parallel and Distributed Systems, 1990.
http://www4.informatik.uni-erlangen.de/~tsthiel/Papers/alewife-lazy-task-creation.ps.gz

[Futures2] http://www.cs.wisc.edu/~fischer/cs538.s01/multilisp.pdf