Consultative Committee for Space Data Systems

RECOMMENDATION FOR SPACE DATA SYSTEM STANDARDS

Registry Repository Reference Model

CCSDS 314.0-W-3

DRAFT WHITE BOOK

January 29, 2011

AUTHORITY

Issue:	
Date:	
Location:	

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS Recommendations is detailed in *Procedures Manual for the Consultative Committee for Space Data Systems*, and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the address below.

This document is published and maintained by:

CCSDS Secretariat
Office of Space Communication (Code M-3)
National Aeronautics and Space Administration
Washington, DC 20546, USA

STATEMENT OF INTENT

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of member space Agencies. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed **Recommendations** and are not considered binding on any Agency.

This **Recommendation** is issued by, and represents the consensus of, the CCSDS Plenary body. Agency endorsement of this **Recommendation** is entirely voluntary. Endorsement, however, indicates the following understandings:

– Whenever an Agency establishes a CCSDS-related **standard**, this **standard** will be in accord with the relevant **Recommendation**. Establishing such a **standard** does not preclude other provisions which an Agency may develop.

– Whenever an Agency establishes a CCSDS-related standard, the Agency will provide other CCSDS member Agencies with the following information:

    - The **standard** itself.
    - The anticipated date of initial operational capability.
    - The anticipated duration of operational service.

– Specific service arrangements are made via memoranda of agreement. Neither this Recommendation nor any ensuing standard is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this **Recommendation** will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or, (3) be retired or canceled.

In those instances when a new version of a **Recommendation** is issued, existing CCSDS-related Agency standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each Agency to determine when such standards or implementations are to be modified. Each Agency is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommendation.

FOREWORD

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommendation is therefore subject to CCSDS document management and change control procedures which are defined in the *Procedures Manual for the Consultative Committee for Space Data Systems*. Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be addressed to the CCSDS Secretariat.

DOCUMENT CONTROL

Document	**Title and Issue**	**Date**	**Status**
0.1	Registry and Repository Reference Model	23-Oct-07	Draft
0.2	Registry and Repository Reference Model	25-Jun-09	Updated to reflect comments – CCSDS Spring Meeting '09
0.3	Registry and Repository Reference Model	02-Feb-10	Updated to reflect comments from CCSDS Fall Meeting '09
0.4	Registry and Repository Reference Model	20-Oct-10	Added reference model and move XML schema extension to annex.
0.5	Registry Reference Model	28-Dec-10	Regenerated API section and added Intrinsic Object section.
0.6	Registry Reference Model	29-Jan-11	Moved Use Case Annexes to separate book; used ontology for API descriptions

CONTENTS

Section Page

Table of Figures

Table of Tables

1 INTRODUCTION

This concept paper represents the beginning of a series of CCSDS Recommendations and Reports meant to provide CCSDS registry/repository recommendations to accommodate the current computing environment and meet evolving requirements.

1.1 PURPOSE AND SCOPE

The main purpose of this document is to define a staged set of CCSDS Recommendations for registries and repositories that meet current CCSDS agency requirements and can be implemented to demonstrate practical, near-term results. This specification needs to be augmented with substantial proof-of-concept and performance prototyping of several registries and repositories in CCSDS environments.

The scope of application of this document is the entire space informatics domain from operational messaging to science archives. In recognition of this varied user community, this document proposes aggressive use of current and emerging standards. In particular, a significant amount of material has been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [7] and the ebXML Registry Information Model (ebXML RIM) [3] specifications.

1.2 BACKGROUND

Registries are pervasive components in most information systems. For example, data dictionaries, service registries, LDAP directory services, and even databases provide "registry-like" services. These all include an account of informational items that are used in large-scale information systems ranging from data values such as names and codes, to vocabularies, services and software components. The focus of this document is the registry/repository, "an information system that securely manages any content type and the standardized **metadata** that describes it." [6] In a registry/repository, the repository is a store for the content. The registry manages the registration of the content.

1.3 STRUCTURE OF THIS DOCUMENT

This document is divided into informative and normative chapters and annexes.

Sections 1- 3 of this document are informative chapters that give a high level view of the rationale, the conceptual environment, some of the important design issues, an overview of registry use cases and an introduction to the terminology and concepts.

- Section 1 gives background to this effort, its purpose and scope, a view of the overall document structure, and the acronym list, glossary, and reference list for this document.
- Section 2 provides a high level view of the anticipated computing environment and the key concepts in the domain of registries and repositories.
- Section 3 provides use case scenarios that convey how actors interact with a registry/repository in the most general cases. These use cases convey a general concept of actors and functions that are supported by registries.

Sections 4 –11 of this document are the normative portion of the specification.

- Section 4 presents a registry reference information model. The information model defines the classes needed to support the essential functions provided by a registry that allows an organization to publish and discover services and artifacts.

- Section 5 presents a facade API that performs registry/repository operations over a diverse set of registries and defines a unified information model for describing registry/repository contents. Regardless of the registry provider, applications use common APIs and a common information model.

- Section 6 presents a federated model that includes features for federated query support, linking of content and metadata across registry boundaries, replication and synchronization of content and metadata among repositories, moving of content and metadata from one registry/repository to another, and event notifications.

- Section 7 presents a model for extrinsic objects,. Since the registry/repository can contain arbitrary content without intrinsic knowledge about that content, the extrinsic object models allows special metadata attributes to provide some knowledge about the object.

- Section 8 presents a model for intrinsic objects. The registry has knowledge about that content of intrinsic registry objects.

- Section 9 defines protocols supported by the Lifecycle Management service interface of the registry/repository. The Lifecycle Management protocols provide the functionality required by RegistryClients to manage the lifecycle of `RegistryObjects` and `RepositoryItems` within the registry.

- Section 9 presents the a reference implementation.

Annexes
- Annex 1 describes Electronic Business using XML Registry
- Annex 2 provides the use cases for an XML Schema Registry, an extension of the generic registry reference model.

1.4 DEFINITIONS

1.4.1 ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CCSDS	Consultative Committee for Space Data Systems
CIO	Content Information Object
CPA	Collaboration Protocol Agreement
CPP	Collaboration Protocol Profile
CORBA	Common Object Request Broker Architecture
DTD	Document Type Definition
ebXML	Electronic Business using eXtensible Markup Language
EJB	Enterprise Java Bean
HTTP	Hypertext Transfer Protocol
ISO	International Organization for Standardization
JAXR	Java API for XML Registries
MIME	Multipurpose Internet Mail Extensions
OAIS	Open Archival Information System
PDS	Planetary Data System
REST	Representational State Transfer
RIM	Registry Information Model
SFDU	Standard Formatted Data Unit
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery & Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Wed Consortium
XFDU	XML Formatted Data Unit
XML	Extensible Markup Language

1.4.2 TERMINOLOGY

CCSDS Control Authority: An organization under the auspices of the CCSDS that supports the transfer and usage or SFDUs by providing operational services of registration, archiving, and dissemination of data descriptions. It is comprised of:

- The CCSDS Secretariat supported by the Control Authority Agent
- Member Agency Control Authority Offices

Choreography: Choreography concerns the interactions of services with their users. Any user of a Web service, automated or otherwise, is a client of that service.

Content Data Object: The Data Object, together with the associated Representation Information, is the original target of preservation.

Content Information: The set of information that is the original target of preservation. It is an Information Object comprised of its Content Data Object and its Representation Information. An example of Content Information could be a single table of numbers representing, and understandable as, temperatures, but excluding the documentation that would explain its history and origin, how it relates to other observations, etc.

Data: A reinterpretable representation of information in a formalized manner suitable for communication, interpretation, or processing. Examples of data include a sequence of bits, a table of numbers, the characters on a page, the recording of sounds made by a person speaking, or a moon rock specimen.

Data Object: Contains some file content and any data required to allow the information consumer to reverse any transformations that have been performed on the object and restore it to the byte stream intended for the original designated community and described by the Representation metadata in the Content Unit

Designated Community: An identified group of potential Consumers who should be able to understand a particular set of information. The Designated Community may be composed of multiple user communities.

Information: Any type of knowledge that can be exchanged. In an exchange, it is represented by data. An example is a string of bits (the data) accompanied by a description of how to interpret a string of bits as numbers representing temperature observations measured in degrees Celsius (the representation information).

Information Object: A Data Object together with its Representation Information.

Metadata Object: Metadata Object in the context of a registry object is a single instance of Representation Information.

Operation: An operation in the context of a service definition can be compared to a function in a traditional programming language.

Orchestration: The orchestration of a service defines how the overall functionality of the service is achieved by the cooperation of other services.

Representation Information: The information that maps a Data Object into more meaningful concepts. An example is the ASCII definition that describes how a sequence of bits (i.e., a Data Object) is mapped into a symbol.

1.5 REFERENCES

[1] Information Architecture Reference Model, CCSDS 312.0-G-1, Draft Green Book, June 2006.

[2] Reference Model for an Open Archival Information System (OAIS), CCSDS 650.0-B-1, Blue Book, January 2002.

[3] OASIS/ebXML Registry Information Model Version 3.0.1, Committee Draft, OASIS/ebXML Registry Technical Committee, February 2007.

[4] Najmi, Farrukh, "Web Content Management Using the OASIS ebXML Registry Standard", XML Europe 2004, http://www.idealliance.org/papers/dx_xmle04/papers/04-02-02/04-02-02.html, April 2004.

[5] Java API for XML Registries (JAXR), JAXR Version 1.0, Sun MicroSystems, 2002.

[6] CCSDS Registry Information Model Specification, Draft White Book, Version 0.080303, September 2008.

[7] ebXML Registry Services and Protocols, Version 3.0, 15 March, 2005

[8] ISO/IEC, " ISO 14721:2003 Open archival information system -- Reference model" 2003, International Organization for Standardization, Geneva, Switzerland.

[9] ISO/IEC, "ISO/IEC 11179:3 Information Technology -- Metadata registries (MDR), 2007-05-27, International Organization for Standardization, Geneva, Switzerland.

[10] CCSDS Registry Use Cases, Draft White Book, Version 0.110129, January, 2011.

2 OVERVIEW OF A REGISTRY/REPOSITORY

This section provides an overview of some of the key concepts that are incorporated in the design of the registry/repository recommendation. A registry addresses the following essential functional requirements:

- Discovery and maintenance of registered content.
- Support for collaborative development, where users can create content and submit it to the registry for use and potential enhancement by the authorized parties.
- Persistence of registered content and science documents.
- Secure version control of registered content.
- Federation of cooperating registries to provide a single view of registered content by seamless querying, synchronization, and relocation of registered content.
- Event notification.

A registry implementation complies with the specification if it meets the following conditions:

- It supports the registry information model.
- It supports the syntax and semantics of the registry interfaces and security.
- It supports the registry schema.

2.1 ENVIRONMENT CONTEXT FOR A REGISTRY/REPOSITORY

Figure 1 illustrates a registry/repository in the context of a generic layered system environment. The registry/repository foundation in the framework includes features for query support, linking of content and metadata, replication and synchronization of content, and event notification. In a federated environment—an environment in which multiple providers agree upon standard operation in a collective fashion—these features extend over the federated registries. An Application Programming Interface (API) supports registry operations over a diverse set of registries and defines a unified information model for describing registry contents. Finally, access control and data management modules, tools, and a governance model bridge the functionality gap to support the enterprise applications.

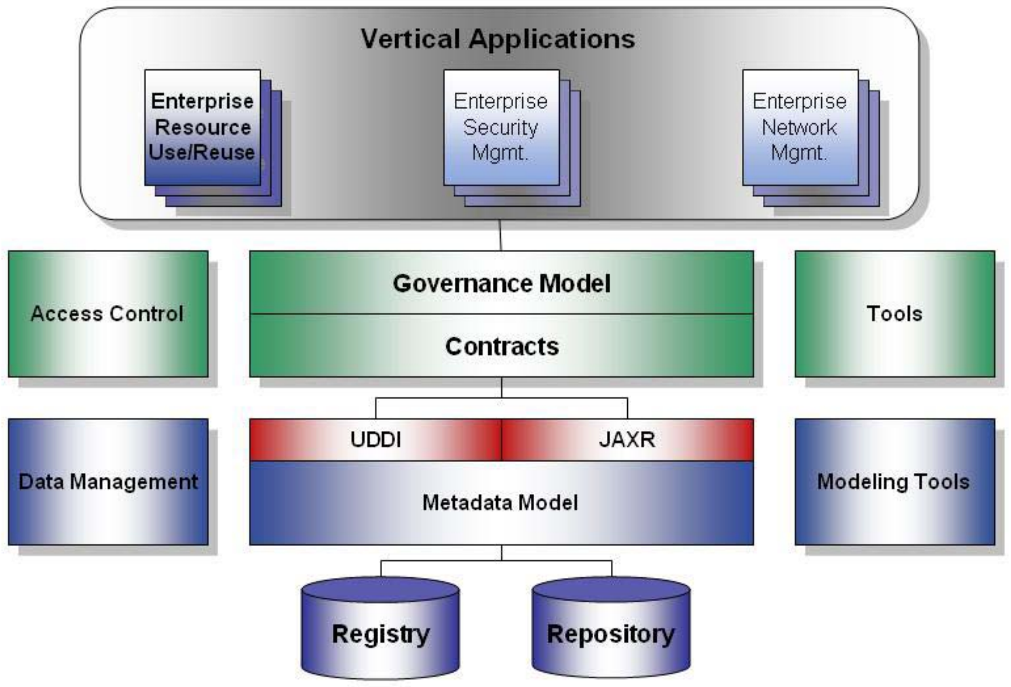


Figure 1: Environment View of a Registry/Repository

2.2 FUNCTIONAL VIEWS OF A REGISTRY/REPOSITORY

A registry/repository allows organizations to publish and discover resources. The two dominate industry standards for registry/repository are Universal Description, Discovery, and Integration (UDDI) and Electronic Business using eXtensible Markup Language (ebXML).

There are two major types of registry/repositories, one that functions as a Service Address Book and the other as an Information Repository. Both are illustrated in Figure 2. As a Service Address Book, the service is first registered. A software element subsequently looks up the service and then executes the service. As an information repository, the software element simply requests then receives the resource.

This document focuses on a generic reference model for a registry/repository, where services and generic information artifacts are managed in the same way, to the greatest degree possible.

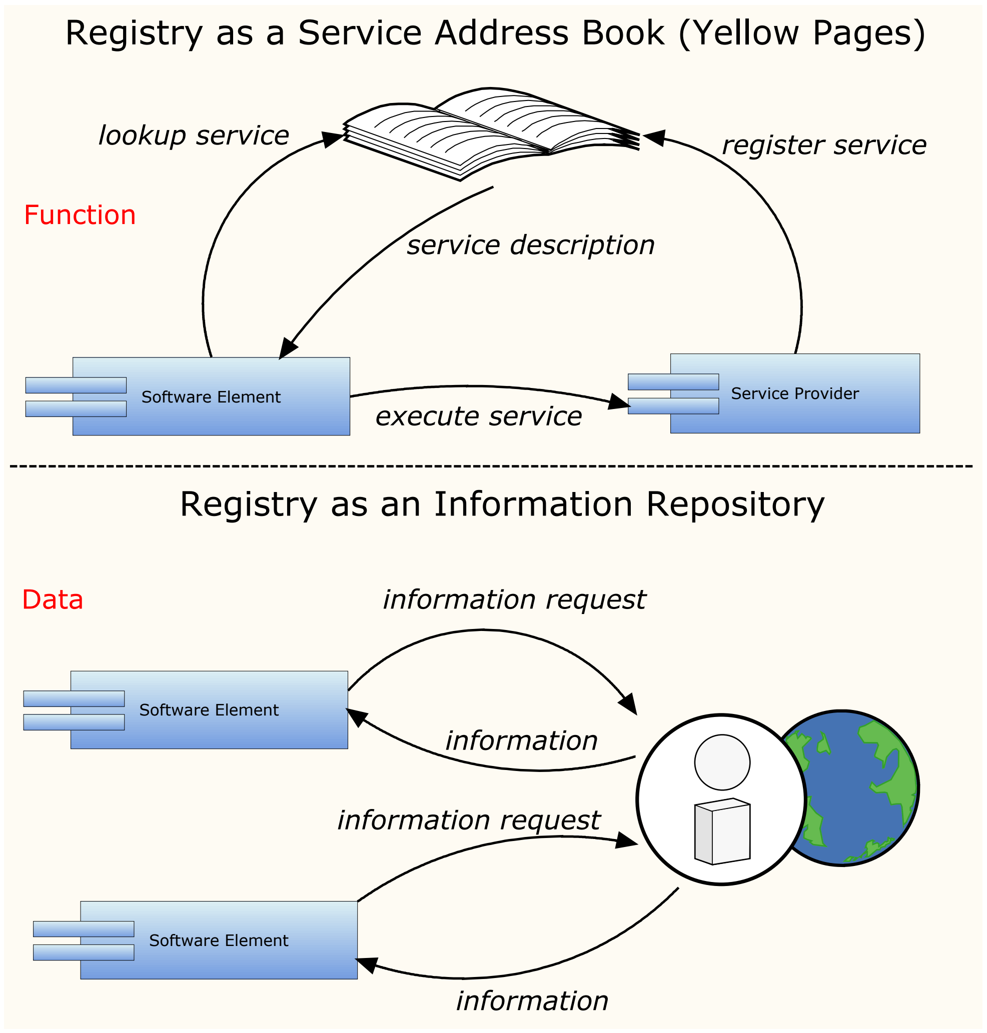


Figure 2: Two Conceptual Views of a Registry/Repository

2.3 GENERAL FEATURES OF A REGISTRY/REPOSITORY

A registry and repository need to support the registration and discovery of information artifacts and services by providing interfaces for their submission, approval, and publishing as well as query capabilities for searching, metadata management capabilities for classification and association, governance and control authorities for maintaining integrity, change control processes for management, and effective access by both people and computer systems. Figure 3 illustrates these features. The API and Information Model sections of this white book describe the proposed API and model classes that support the functionalities associated with Content Management, Events, Secure Architecture, and Services Registry. The Federation section of this document describes the Federated Architecture.

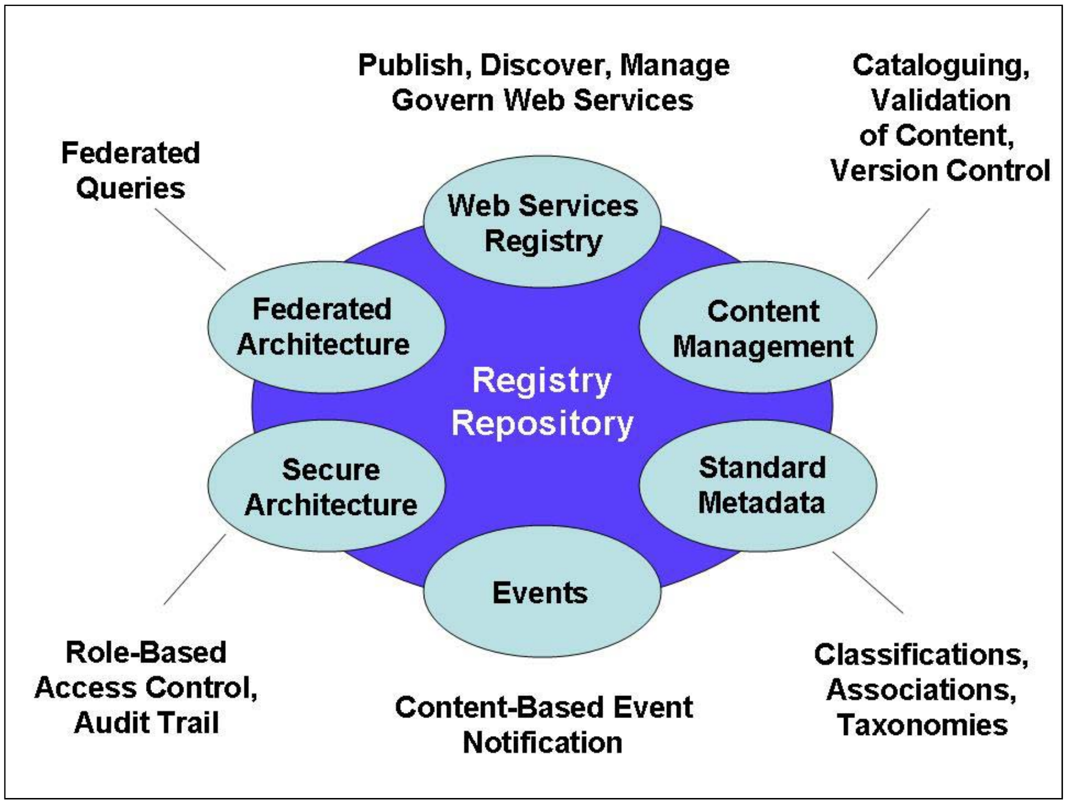

Publish, Discover, Manage
Govern Web Services
Cataloguing, Validation of Content, Version Control
Federated Queries
Web Services Registry
Federated Architecture
Content Management
Registry Repository
Secure Architecture
Standard Metadata
Events
Role-Based Access Control, Audit Trail
Content-Based Event Notification
Classifications, Associations, Taxonomies

Figure 3: Features of a Federated Registry/Repository

3 USE CASES

The purpose of this section is to capture use case scenarios[1] for registries and repositories. These use cases have been derived from several sources including the CCSDS Reference Architecture for Space Data Systems (RASIM) [1], and formally defined Use Cases for a number of projects, including, for example, the Deep Space Mission System (DSMS) Information Services Architecture Registry [7], Open Geographic Information Systems (GIS) Project Registry, ASAR MERIS AATSR Labeling Facility Inspection (AMALFI) Multi-Missions – Xml Schema Repository, JPL Deep Space Network (DNS) Information Service Architecture Registry, CCSDS Service Link Exchange (SLE) Working Group (WG), CCSDS Navigation WG, Common and Core Spacecraft Monitor & Control (SM&C), Operations Automation and Scheduling, Remote Software Management, Payload Data Product Management, and Operator Interaction. The detailed uses cases have been compiled into a separate document. [10]

3.1 ACTORS

The following actors[2] are identified for the registry/repository use cases.

Publisher «system or person» - A publisher is a system or person that provides an artifact to be submitted into the registry/repository.

Artifact Consumer «system or person» - An artifact consumer is a system or person that receives an artifact from a registry/repository.

Subscriber «system or person» - A subscriber is a person or system that has the right to receive a notification about a change in status of an artifact in a registry/repository.

System Administrator «person» - A system administrator is a person employed to maintain, and operate a registry/repository configuration.

Registry/Repository Service «system» - A registry/repository service is a system interface through which another actor is able to perform registry/repository functions which include changes to the catalog and/or repository.

[1] A *use case* is a set of scenarios tied together by a common user goal. A *scenario* is a sequence of steps describing an interaction between a user and a system [M. Fowler, *UML Distilled*, Third Ed., Addison-Wesley, 2004].

[2] An actor is a role that a user plays with respect to the system, external to the system, attempting to achieve a goal. Actors can be human or non-human (e.g., other systems) that carry out or support use cases [Ibid].

3.2 GENERAL USE CASES

This section provides use case scenarios that convey how actors interact with a registry/repository in the most general cases. Since general use cases have been developed by many other tasks such as those cited earlier, the intent here is to convey a general concept of actors and functions that are supported by registries/repositories.

Figure 4 illustrates a generalized view of the interactions between actors and registry/repository services, the interface available for performing registry/repository functions.

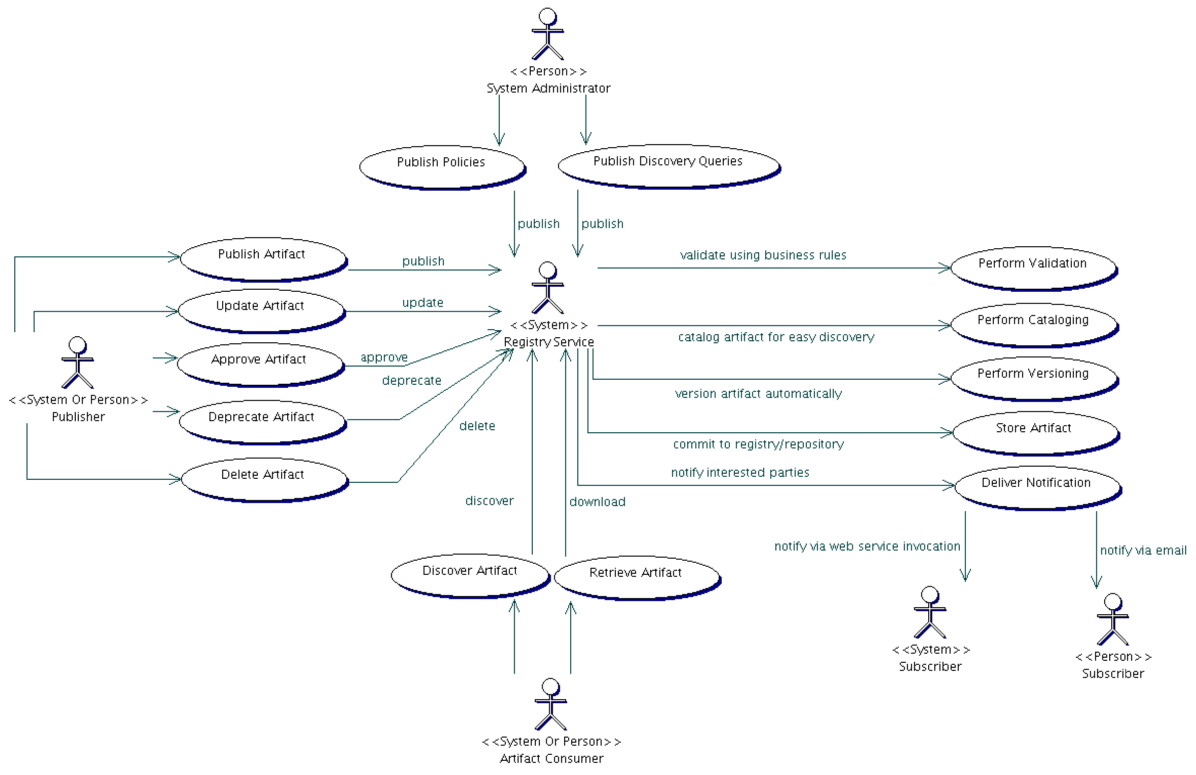


Figure 4: General Registry/Repository Use Cases

3.2.1 PUBLISH

Description: This use case describes the actions necessary for a user to publish an artifact in the registry/repository.

Actors: Publisher

Actions:

1. Publisher publishes a new artifact in the registry which includes descriptive metadata about the artifact

2. Registry/Repository Service validates the metadata (include Validate use case)

3. Registry/Repository Service assigns a version identifier for the artifact (include Version use case)

4. Registry/Repository Service updates the catalog with the metadata (include Catalog use case)

5. Registry/Repository Service stores the artifact in the repository (include Store use case)

6. Registry/Repository Service returns an identifier and version for the published artifact

7. Registry/Repository Service sends notification to the subscribers regarding the published artifact (include Notify use case)

3.2.2 UPDATE

Description: This use case describes the actions necessary to update an artifact in the registry/repository.

Actors: Publisher

Actions:

1. Publisher requests that an artifact be updated based on an identifier and version for the artifact

2. Registry/Repository Service replaces the artifact in the repository (include Store use case)

3. Registry/Repository Service updates the metadata for the artifact in the catalog (include Catalog use case)

4. Registry/Repository Service sends notification to the subscribers updated artifact (include Notify use case)

3.2.3 APPROVE

Description: This use case describes the actions necessary to approve an artifact in the registry/repository.

Actors: Publisher

Actions:

1. System Administrator queries the registry/repository service for newly published artifacts which are not already approved
2. System Administrator updates the metadata for an artifact to indicate that it is approved
3. Registry/Repository Service sends notification to the publisher and subscribers of the approved artifact (include Notify use case)

3.2.4 DEPRECATE

Description: This use case describes the actions necessary to deprecate an artifact in the registry/repository. This is normally triggered by the registration of a new artifact with the logical identifier of an existing artifact but with a new version.

Actors: Publisher

Actions:

1. System Administrator updates the registry/repository to indicate that a specific artifact and version is deprecated
2. Registry/Repository Service sends notification to the subscribers of the deprecated artifact (include Notify use case)

3.2.5 UNDEPRECATE

Description:

This use case describes the actions necessary to undeprecate an artifact in the registry/repository. This use case is not pictured in Figure 4.

Actors: Publisher

Actions:

1. System Administrator updates the registry/repository to indicate that a specific artifact and version is undeprecated
2. Registry/Repository Service sends notification to the subscribers of the undeprecated artifact (include Notify use case)

3.2.6 DELETE

Description: This use case describes the actions necessary to delete an artifact in the registry/repository.

Actors: Publisher

Actions:

1. Publisher requests that an artifact be deleted based on an identifier for the artifact
2. Registry/Repository Service deletes the artifact from the repository
3. Registry/Repository Service removes the artifact from the catalog
4. Registry/Repository Service sends notification to the subscribers of the deleted artifact (include Notify use case)

3.2.7 VALIDATE

Description: This use case describes the actions necessary to validate the metadata associated with an artifact. This use case is included as part of the Publish use case.

Actors: Registry/Repository Service

Actions:

1. Registry/Repository Service validates the metadata associated with the artifact

3.2.8 CATALOG

Description: This use case describes the actions necessary to catalog a new artifact. This use case is included as part of the Publish and Update use cases.

Actors: Registry/Repository Service

Actions:

1. Registry/Repository Service updates the catalog with the metadata

3.2.9 VERSION

Description: This use case describes the actions necessary to version a new artifact. This use case is included as part of the Publish use case.

Actors: Registry/Repository Service

Actions:

1. Registry/Repository Service assigns a version identifier to the artifact

3.2.10 STORE

Description: This use case describes the actions necessary to store the artifact. This use case is included as part of the Publish and Update use cases.

Actors: Registry/Repository Service

Actions:

1. Registry/Repository Service stores the artifact in the repository

3.2.11 NOTIFY

Description: This use case describes the actions necessary to notify subscribers of registry/repository events. This use case is included as part of the Publish, Update, Approve and Deprecate use cases.

Actors: Registry/Repository Service, Subscriber

Actions:

1. Registry/Repository Service notifies the subscriber when an event of interest has occurred

2. Subscriber receives the notification

3.2.12 DISCOVER

Description: This use case describes the actions necessary to discover registered artifacts

Actors: Artifact Consumer

Actions:

1. Artifact Consumer enters search criteria
2. Registry/Repository Service searches the catalog and returns metadata, including location information, describing registered artifacts that meet the search criteria.

3.2.13 RETRIEVE

Description: This use case describes the actions necessary to retrieve registered artifacts.

Actors: Artifact Consumer

Actions:

1. Artifact Consumer enters an identifier for the artifact
2. Registry/Repository Service retrieves the artifact from the repository and returns the artifact in its original form

3.3 ADMINISTRATION USE CASES

The administration use cases describe the actions necessary to manage the registry/repository. These use cases include user management, system management and policy management. Section 8, Lifecycle Management, addresses many of these use cases.

3.4 SPECIFIC USE CASES

This section provides use case scenarios for specific subclasses of registries. These are extensions to the general set of registry/repository use cases described in Section 3.2. They are specific to a Service Registry. They are specific to a Service Registry. Use cases for an XML Schema registry are in Annex 2.

3.4.1 SERVICE REGISTRY/REPOSITORY USE CASES

Service registries allow for the registration of services, in particular, for service-oriented architectures. Service registries inherit several of the functions of a general registry/repository allowing for registration and discovery of online services.

3.4.1.1 Actors

Service provider - A service provider is a system or person that provides a service to be submitted into the registry/repository. This actor is equivalent to the Publisher defined in Section 3.1.

Service consumer - A service consumer is a system or person that discovers and receives descriptions of services in the registry/repository. This actor is equivalent to the Artifact Consumer defined in Section 3.1.

3.4.1.2 Service Registration

Description: The service provider adds information about the service to the registry.

Actors: Service Provider

Scenarios:

1. A Service Provider registers a service through a registry/repository service using standardized metadata.

2. The registry/repository service adds the metadata describing the service to the catalog.

3. The registry/repository service allows classification of the registered service based on namespace.

4. The registry/repository service allows the association of services with other related resources (URIs, web sites, documentation, etc) located both locally and externally.

3.4.1.3 Service Discovery

Description: A service consumer requests the service information from the registry/repository.

Actors: Service Consumer

Scenarios

1. A Service Consumer requests information about registered services from the service registry/repository using query parameters.

2. The registry/repository service returns information that corresponds to registered services based on the specific attributes passed to the service. This includes information related to accessing and connecting to the service.

3.4.1.4 Service Removal

Description: A service provider requests the service be removed from the registry/repository.

Actors: Service Provider

Scenarios

1. A Service Provider requests the service registry/repository to remove the service.

2. The registry/repository service removes the entry from the catalog.

4 INFORMATION MODEL

The CCSDS Registry Reference Information Model (RIM) is exactly the ebXML Registry Information Model (RIM) [3]. This model typically results from registry/repository design efforts that support the registry/repository use cases presented above. For example the RIM is the assumed registry/repository model for JAXR (Java API for XML Registries). The ebXML RIM subsumes the UDDI information model and so includes modeled components for a service registry.

4.1 OVERVIEW OF A REGISTRY/REPOSITORY INFORMATION MODEL

A registry/repository allows organizations to publish and discover resources. The registry/repository information model defines the classes and associations that support the registry/repository features illustrated in Figure 3. The objects acted on by the registry/repository API are defined in the information model and support the required functionality such as publishing, discovery and management of registry/repository objects.

The CCSDS registry/repository information model provides a blueprint or high-level schema for a CCSDS registry/repository. It provides implementers with information on the type of metadata that is stored in the registry/repository as well as the relationships among metadata classes. The registry/repository information model defines what types of objects are stored and organized in the registry/repository.

Extensions are allowed to the model and should be based on existing standards. For example, an observational product registry would be based on the Open Archive Information System (OAIS) [8] and the ISO 11179 [9] Registry/Repository models.

4.2 VIEWS OF THE REGISTRY/REPOSITORY MODEL

The CCSDS Registry/Repository Reference Information Model provides a formal data engineering definition of the registry/repository information model and is exactly the ebXML Registry Information Model (RIM) specification. [3] In the following two sections high-level conceptual and logical views of the information model are provided.

4.2.1.1 Conceptual

A conceptual model defines the community model from a manager's point of view and is concerned with the language of the community, mainly concepts, facts, words, and symbols. Some key concepts are listed in the following table. These concepts are then presented in the concept map in Figure 5.

Registry Components	**Registry Function**
Registry, Registry Object, Registry Package, Classification	Discovery and maintenance of registered content.
Identifiable, Version Information, Auditable Event, Service	Support for collaborative development, where users can create content and submit it to the registry for use and potential enhancement by the authorized parties..
Registry Package	Persistence of registered content and science documents.
Version Information, Auditable Event	Secure version control of registered content.
Federation, External Identifiers, Service, Classification	Federation of cooperating registries to provide a single view of registered content by seamless querying, synchronization, and relocation of registered content.
Auditable Event	Event notification.

Table 1: CCSDS Registry/Repository Components and Functions

The following Conceptual Map illustrates key information model concepts and their relationships.

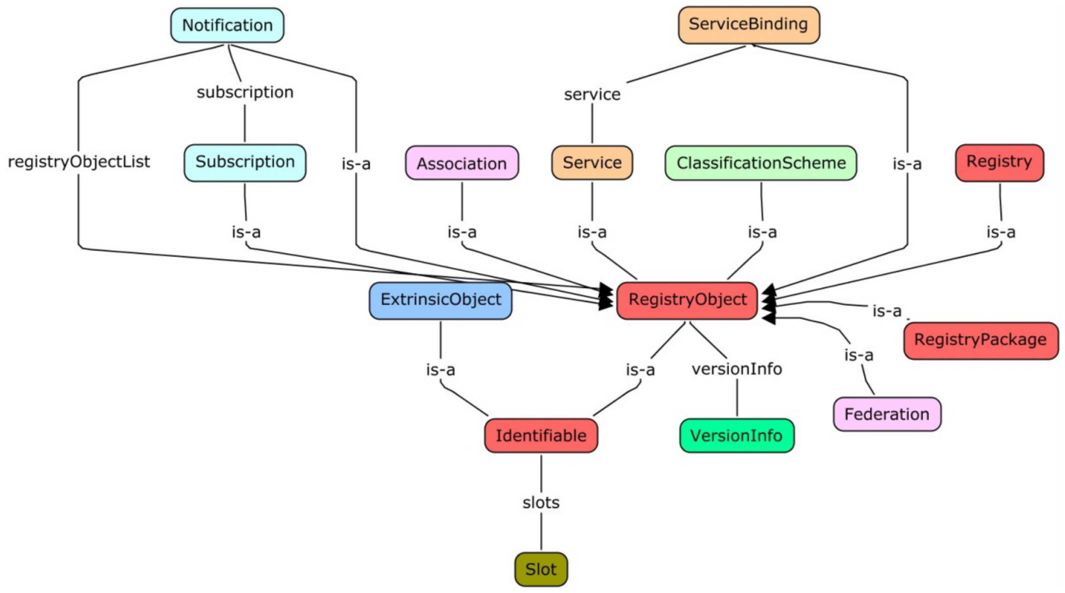


Figure 5: CCSDS Registry/Repository Conceptual Map - Key Classes

4.2.1.2 Logical

The logical model defines the system model of data from a designer's point of view and is concerned with entity classes, attributes, and relationships that describe the things of significance in rigorous terms. Figure 6 illustrates the logical model for the key registry/repository classes.

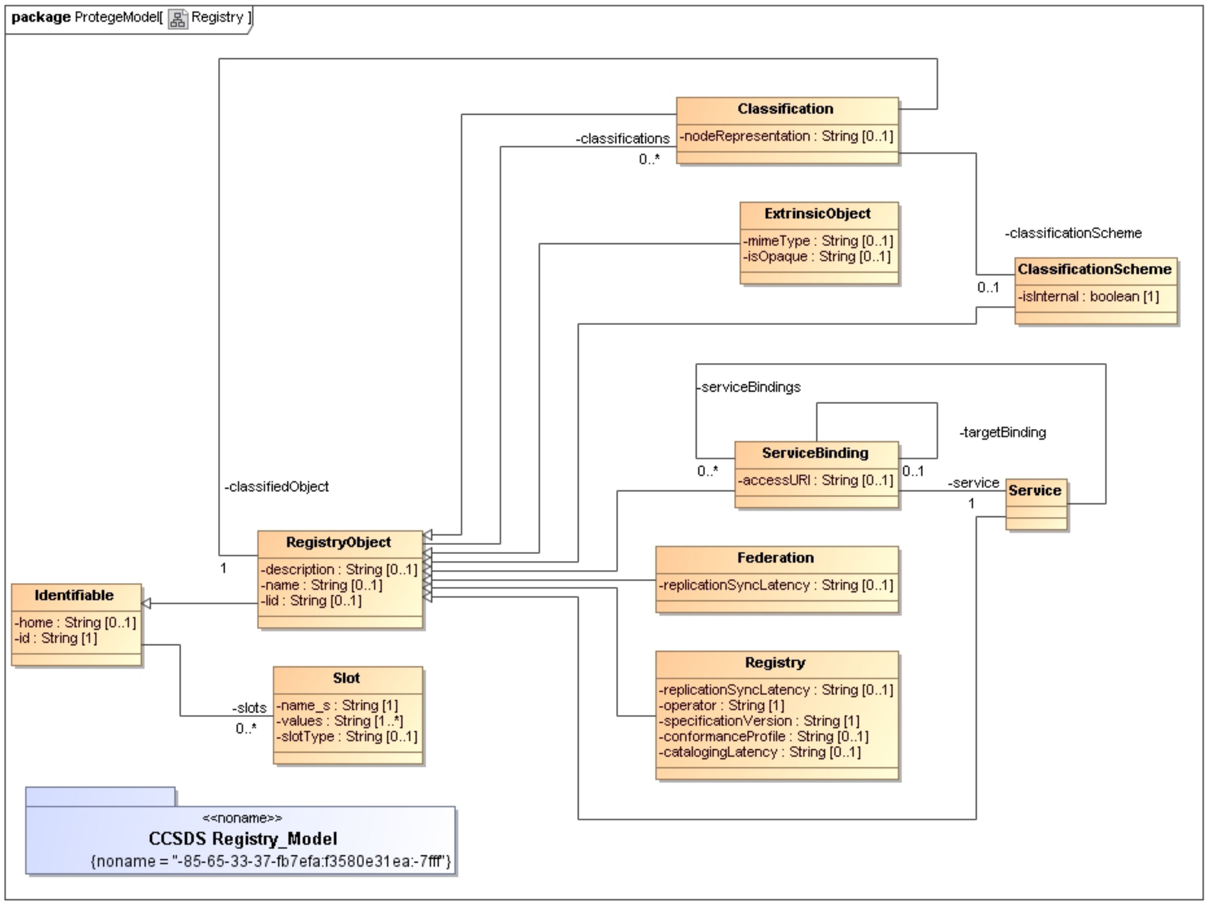

package ProtegeModel[Registry]
Classification
-nodeRepresentation : String [0..1]
-classifications
0..*
ExtrinsicObject
-mimeType : String [0..1]
-isOpaque : String [0..1]
-classificationScheme
ClassificationScheme
-isInternal : boolean [1]
0..1
-serviceBindings
-targetBinding
ServiceBinding
-accessURI : String [0..1]
0..*
0..1
-service
1
Service
-classifiedObject
RegistryObject
-description : String [0..1]
-name : String [0..1]
-lid : String [0..1]
1
Identifiable
-home : String [0..1]
-id : String [1]
Federation
-replicationSyncLatency : String [0..1]
Registry
-replicationSyncLatency : String [0..1]
-operator : String [1]
-specificationVersion : String [1]
-conformanceProfile : String [0..1]
-catalogingLatency : String [0..1]
Slot
-name_s : String [1]
-values : String [1..*]
-slotType : String [0..1]
-slots
0..*
<<noname>>
CCSDS Registry_Model
{noname = "-85-65-33-37-fb7efa:f3580e31ea:-7fff"}

Figure 6: Key Registry/Repository Class Definitions

5 APPLICATION PROGRAMMING INTERFACE (API)

5.1 OVERVIEW

This specification defines a façade[3] API for a client that assumes an underlying services model. Service(s) provide an interface to the underlying implementation and hides the API of the implemented registry, for example, the Java™ API for XML Registries (JAXR) API for an ebXML registry/repository. This façade API should map to the registry/repository use cases described in Section 3. The following figure illustrates the façade API that hides the diverse registry APIs.

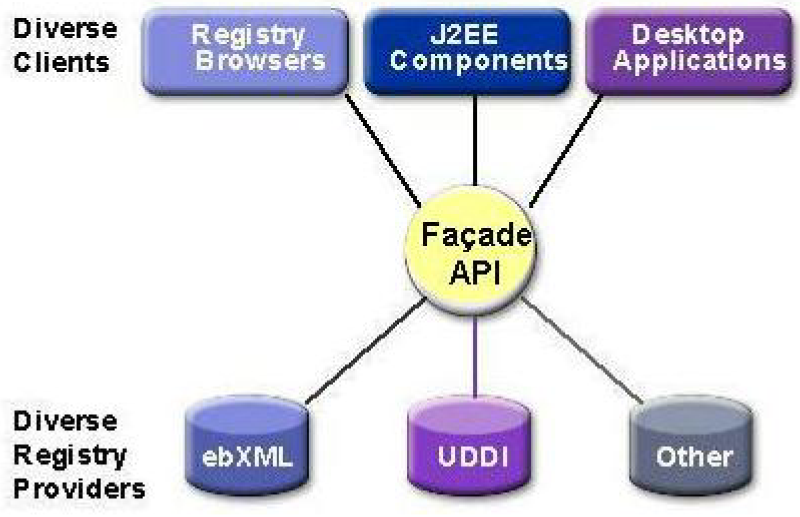


Figure 7: Façade API between Clients and Registries

5.2 CAPABILITY PROFILES

Because some diversity exists among registry provider capabilities, a multilayer API abstraction is offered through *capability profiles*. Each method of the interface is assigned a capability level, and those methods with the same capability level define the provider capability profile.

Currently, two capability profiles are defined: level 0 profile for basic features and level 1 profile for advanced features. Level 0's basic features support so-called business-focused

[3] A façade defines a higher level interface that makes the registries easier to use.

APIs, while level 1's advanced features support generic APIs. At the minimum, all providers must implement a level 0 profile. A client application using only those methods of the level 0 profile can access any provider in a portable manner. For example, the JAXR APIs that support UDDI registries are level 0. The JAXR APIs that support ebXML registries are level 1. The façade API proposed below, based on the registry/repository use cases, can be considered level 2.

5.3 FAÇADE API DEFINITIONS

The following table presents each API as an abstract service. Each abstract service is defined as having a set of operations with pre- and post- conditions. Choreography is the interaction between a user and one or more service required to accomplish the goal. Orchestration identifies the services required to accomplish the goals.

Service	***Description***
Publish_Registry_Object	The Publish_Registry_Object service performs all actions necessary to fully register a registry object in the registry.
PreCondition	Registry_Object_Not_Registered
Operation	Assign_Unique_Identifier
	Assign_User_Logical_Identifier
	Validate_Registry_Object_Metadata
	Add_Catalog_Entry
	Store_Data_Object_To_Repository
	Store_Metadata_Object_To_Repository
	Set_Registry_Object_Status_Pending
	Notify_Subscribers
Choreography	Operation:Get_Metadata_Object_From_User
	Operation:Get_Data_Object_From_User
Orchestration	Service:Publish_Registry_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Data_Object_In_Repository
	Catalog_Entry_Exists

Publish_Registry_Object _No_Data	The Publish_Registry_Object_No_Data service performs all actions necessary to fully register an object in the registry. However, no data object is placed in the repository.
PreCondition	Registry_Object_Not_Registered
Operation	Assign_Unique_Identifier

	Assign_User_Logical_Identifier
	Validate_Registry_Object_Metadata
	Add_Catalog_Entry
	Store_Metadata_Object_To_Repository
	Set_Registry_Object_Status_Pending
	Notify_Subscribers
Choreography	Operation:Get_Metadata_Object_From_User
	Service:Publish_Registry_Object_No_Data
Orchestration	Service:Publish_Registry_Object_No_Data
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Catalog_Entry_Exists

Update_Registry_Object	The Update_Registry_Object service replaces a registry object's metadata and data object in the repository and updates the registry.
PreCondition	Registry_Object_Registered
Operation	Find_Registry_Object
	Remove_Data_Object_From_Repository
	Remove_Metadata_Object_From_Repository
	Update_Catalog_Entry
	Store_Data_Object_To_Repository
	Store_Metadata_Object_To_Repository
	Set_Registry_Object_Status_Pending
	Notify_Subscribers
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Update_Registry_Object
Orchestration	Service:Update_Registry_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Data_Object_In_Repository
	Catalog_Entry_Exists

Approve_Registry_Object	The Approve_Registry_Object service sets the registry object status to APPROVED.
PreCondition	Registry_Object_Registered
Operation	Find_Registry_Object
	Set_Registry_Object_Status_Approved
	Notify_Subscribers
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Approve_Registry_Object

Orchestration	Service:Approve_Registry_Object
PostCondition	Current_Registry_Object_Available

Deprecate_Registry_Object	The Deprecate_Registry_Object service sets the status of the registry object to DEPRECATED.
PreCondition	Registry_Object_Registered
Operation	Find_Registry_Object
	Notify_Subscribers
	Set_Registry_Object_Status_Deprecated
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Deprecate_Registry_Object
Orchestration	Service:Deprecate_Registry_Object
PostCondition	Current_Registry_Object_Available

Undeprecate_Registry_Object	The Undeprecate_Registry_Object service sets the status of the registry object to UNDEPRECATED.
PreCondition	Registry_Object_Registered
Operation	Find_Registry_Object
	Set_Registry_Object_Status_Pending
	Notify_Subscribers
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Undeprecate_Registry_Object
Orchestration	Service:Undeprecate_Registry_Object
PostCondition	Current_Registry_Object_Available

Delete_Registry_Object	The Delete_Registry_Object service removes the regisry object's metadata and data object from the repository and removes the catalog entry.
PreCondition	Registry_Object_Registered
Operation	Find_Registry_Object
	Remove_Data_Object_From_Repository
	Remove_Metadata_Object_From_Repository
	Remove_Catalog_Entry
	Notify_Subscribers
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Delete_Registry_Object
Orchestration	Service:Delete_Registry_Object
PostCondition	Current_Registry_Object_NotAvailable

Validate_Registry_Object	The Validate_Registry_Object service validates the registry object's metadata.
PreCondition	Registry_Object_Not_Approved
Operation	Find_Registry_Object
	Validate_Registry_Object_Metadata
	Set_Registry_Object_Status_Validated
	Notify_Subscribers
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Validate_Registry_Object
Orchestration	Service:Validate_Registry_Object
PostCondition	Current_Registry_Object_Available
Catalog_Registry_Object	The Catalog_Registry_Object service adds a catalog entry for the registry object.
PreCondition	Registry_Object_Not_Registered
Operation	Add_Catalog_Entry
	Set_Registry_Object_Status_Pending
	Notify_Subscribers
Choreography	Service:Catalog_Registry_Object
	Operation:Return_Status
Orchestration	Service:Catalog_Registry_Object
PostCondition	Current_Registry_Object_Available
Version_Registry_Object	The Version_Registry_Object service publishes a new version of an existing registry object.
PreCondition	Registry_Object_Verrsion_Pre-exists
Operation	Find_Last_Registry_Object
	Set_Registry_Object_Status_Pending
Choreography	Operation:Get_Registry_Object_Logical_Id_From_User
	Service:Version_Registry_Object
Orchestration	Service:Publish_Registry_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Data_Object_In_Repository
	Catalog_Entry_Exists
Store_Data_Object	The Store_Data_Object service places the registry object's data object in the repository.
PreCondition	Registry_Object_Not_Approved
Operation	Find_Registry_Object
	Store_Data_Object_To_Repository

	Set_Registry_Object_Status_Pending
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Operation:Get_Data_Object_From_User
Orchestration	Service:Store_Data_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Data_Object_In_Repository
	Catalog_Entry_Exists
Notify_Subscribers	The Notify_Subscribers service sends a message to subscribers that have requested to be notified of the occuring event.
Operation	Find_Event_Subscribers
	Notify_Subscribers
Choreography	Service:Notify_Subscribers
	Operation:Return_Status
Orchestration	Service:Notify_Subscribers
Find_Registry_Object	The Find_Registry_Object service locates a registry object using unique identifiers.
PreCondition	Registry_Object_Registered
Operation	Find_Registry_Object
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Find_Registry_Object
Orchestration	Service:Find_Registry_Object
PostCondition	Current_Registry_Object_Available
Discover_Registry_Object	The Discover_Registry_Object service locates a registry object using search criteria provided by a user.
PreCondition	Registry_Object_Registered
Operation	Search_Registry_Object
Choreography	Operation:Get_Search_Parameters_From_User
	Service:Discover_Registry_Object
Orchestration	Service:Discover_Registry_Object
PostCondition	PostCondition_Current_Registry_Object_Set_Available
Retrieve_Data_Object	The Retrieve_Data_Object service returns the registry object's data object from the repository.
PreCondition	Registry_Object_Registered

Operation	Find_Current_Registry_Object
	Find_Data_Object_In_Repository
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Retrieve_Data_Object
Orchestration	Service:Retrieve_Data_Object
PostCondition	Current_Registry_Object_Available

Retrieve_Metadata_Object	The Retrieve_Metadata_Object service returns the registry object's descriptive metadata from the repository.
PreCondition	Registry_Object_Registered
Operation	Find_Current_Registry_Object
	Find_Metadata_Object_In_Repository
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Service:Retrieve_Metadata_Object
Orchestration	Service:Retrieve_Metadata_Object
PostCondition	Current_Registry_Object_Available

Set_Registry_Object_Status	The Set_Registry_Object_Status service changes the status of a registry object.
PreCondition	Registry_Object_Registered
Operation	Set_Registry_Object_Status
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Operation:Get_Registry_Object_Status
Orchestration	Service:Set_Registry_Object_Status

Set_Registry_Object_Status_Pending	The Set_Registry_Object_Status_Pending service changes the status of a registry object to pending.
PreCondition	Registry_Object_Registered
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Operation:Get_Registry_Object_Status
Orchestration	Service:Set_Registry_Object_Status

Set_Registry_Object_Status_Validated	The Set_Registry_Object_Status_Validated service changes the status of a registry object to validated.
PreCondition	Registry_Object_Registered
Choreography	Operation:Get_Registry_Object_Identifiers_From_User

	Operation:Get_Registry_Object_Status
Orchestration	Service:Set_Registry_Object_Status

Set_Registry_Object_Status_Approved	The Set_Registry_Object_Status_Approved service changes the status of a registry object to approved.
PreCondition	Registry_Object_Registered
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Operation:Get_Registry_Object_Status
Orchestration	Service:Set_Registry_Object_Status

Set_Registry_Object_Status_Deprecated	The Set_Registry_Object_Status_Deprecated service changes the status of a registry object to deprecated.
PreCondition	Registry_Object_Registered
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Operation:Get_Registry_Object_Status
Orchestration	Service:Set_Registry_Object_Status

Find_All_Associated_Object	The Find_All_Associated_Object service finds all objects in an association to a given object.
PreCondition	Registry_Object_Registered
Operation	Find_Current_Registry_Object
	Find_Associated_Registry_Object
Choreography	Operation:Get_Registry_Object_Identifiers_From_User
	Operation:Get_Association_From_User
Orchestration	Service:Find_All_Associated_Object
PostCondition	Current_Registry_Object_Available

Associate_From_Object	The Associate_From_Object service links a registry object to an association. The registry object is the "from" object in a one directional association.
PreCondition	Registry_Object_Registered
Operation	Find_Current_Registry_Object
	Find_Registry_Object(Association)
	Assign_From_Association
Choreography	Operation:Get_Association_From_User
	Operation:Get_Registry_Object_Identifiers_From_User
Orchestration	Service:Associate_From_Object

PostCondition	Current_Registry_Object_Available
Associate_To_Object	The Associate_To_Object service links a registry object to an association. The registry object is the "to" object in a one directional association.
PreCondition	Registry_Object_Registered
Operation	Find_Current_Registry_Object
	Find_Associationed_Registry_Object(Association)
	Find_Registry_Object
	Assign_To_Association
Choreography	Operation:Get_Association_From_User
	Operation:Get_Registry_Object_Identifiers_Fro m_User
Orchestration	Service:Associate_To_Object
PostCondition	Current_Registry_Object_Available
Deassociate_From_Object	The Deassociate_From_Object service removes the link between a registry object and an association. The registry object is the "from" object in a one directional association.
PreCondition	Registry_Object_Registered
Operation	Find_Current_Registry_Object
	Delete_From_Association
Choreography	Operation:Get_Registry_Object_Identifiers_Fro m_User
	Operation:Get_Association_From_User
Orchestration	Service:Deassociate_From_Object
PostCondition	Current_Registry_Object_NotAvailable
Deassociate_To_Object	The Deassociate_To_Object service removes the link between a registry object and an association. The registry object is the "to" object in a one directional association.
PreCondition	Registry_Object_Registered
Operation	Find_Current_Registry_Object
	Find_Associationed_Registry_Object(Association)
	Delete_To_Association
Choreography	Operation:Get_Registry_Object_Identifiers_Fro m_User
	Operation:Get_Association_From_User
Orchestration	Service:Deassociate_To_Object
PostCondition	Current_Registry_Object_Available

Classify_Object	The Classify_Registry_Object service associates registry object with a classification.
PreCondition	Registry_Object_Registered
Operation	Find_Associationed_Registry_Object(Classification)
	Assign_From_Association
	Find_Current_Registry_Object
	Assign_To_Association
Choreography	Operation:Get_Registry_Object_Id_From_User
	Operation:Get_Classification_Node_From_User
Orchestration	Service:Classify_Object
PostCondition	Current_Registry_Object_Available
Publish_Association	The Publish_Association service performs all actions necessary to fully register a Association in the registry.
PreCondition	Registry_Object_Not_Registered
Choreography	Operation:Get_Metadata_Object_From_User
	Service:Publish_Registry_Object
Orchestration	Service:Publish_Registry_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Catalog_Entry_Exists
Publish_Classification	The Publish_Classification service performs all actions necessary to fully register a Classification in the registry.
PreCondition	Registry_Object_Not_Registered
Choreography	Operation:Get_Metadata_Object_From_User
	Service:Publish_Registry_Object
Orchestration	Service:Publish_Registry_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Catalog_Entry_Exists
Publish_Federation	The Publish_Federation service performs all actions necessary to fully register a Federation in the registry.
PreCondition	Registry_Object_Not_Registered
Choreography	Operation:Get_Metadata_Object_From_User
	Service:Publish_Registry_Object
Orchestration	Service:Publish_Registry_Object

PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Catalog_Entry_Exists
Publish_AuditableEvent	The Publish_Auditable_Event service performs all actions necessary to fully register an AuditableEvent in the registry.
PreCondition	Registry_Object_Not_Registered
Choreography	Operation:Get_Metadata_Object_From_User
	Service:Publish_Registry_Object
Orchestration	Service:Publish_Registry_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Catalog_Entry_Exists
Publish_Notification	The Publish_Notification service performs all actions necessary to fully register a Notification in the registry.
PreCondition	Registry_Object_Not_Registered
Choreography	Operation:Get_Metadata_Object_From_User
	Service:Publish_Registry_Object
Orchestration	Service:Publish_Registry_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Catalog_Entry_Exists
Publish_Subscriber	The Publish_Subscriber service performs all actions necessary to fully register a Subscriber in the registry.
PreCondition	Registry_Object_Not_Registered
Choreography	Operation:Get_Metadata_Object_From_User
	Service:Publish_Registry_Object
Orchestration	Service:Publish_Registry_Object
PostCondition	Current_Registry_Object_Available
	Metadata_Object_In_Repository
	Catalog_Entry_Exists
Join_Federation	The Join_Federation service creates a "federation" object in two registry, making a federation of the two registries.
PreCondition	Registry_Object_Registered
Choreography	Operation:Get_Metadata_Object_From_User
	Service:Join_Federation
Orchestration	Service:Join_Federation

PostCondition	Current_Registry_Object_Available

5.4 SERVICE ITEM DEFINITIONS

The following table provides definition for the items presented in the services table above. These items include operations and pre- and post- conditions.

Item	Description
Add_Catalog_Entry	The Add_Catalog_Entry operation adds an entry to the registry catalog for a registry object.
Assign_From_Association	The Assign_From_Association operation assigns a named Association to the "from" registry object.
Assign_To_Association	The Assign_To_Association operation assigns a named Association to a "to" registry object.
Assign_Unique_Identifier	The Assign_Unique_Identifer operation generates and assigns a unique object identifier and version identier to a regisry object.
Assign_User_Logical_Identifier	The Assign_User_Logical_Identifier operation assignes a user provided logical identifier (without version)
Catalog_Entry_Exists	The Catalog_Entry_Exists post condition indicates that the a catalog entry has been made in for the registry object in the registries catalog.
Current_Registry_Object_Available	The Current_Registry_Object_Available post condition indicates that a registry object has been located and its metadata object and data object are available.
Current_Registry_Object_NotAvailab le	The Current_Registry_Object_NotAvailable post condition indicates that no registry object has been located. No further operations are allowed.
Data_Object_In_Repository	The Data_Object_In_Repository post condition indicates the registry object's data object has been successfully stored in the repository.
Delete_From_Association	The Delete_From_Association operation removes a named Association from the "from" registry object.
Delete_To_Association	The Delete_To_Association operation removes a named Association from a "to" registry object.
Find_Associated_Registry_Object	The Find_Associated_Registry_Object operation locates a registry object using a named association and the current registry object.
Find_Associationed_Registry_Object (Association)	The Find_Associationed_Registry_Object(Associatio n) operation finds an Association registry object

	using a names association and the current registry object.
Find_Associationed_Registry_Object (Classification)	The Find_Associationed_Registry_Object(Classification) operation finds an Classification registry object using a names association and the current registry object.
Find_Current_Registry_Object	The Find_Current_Registry_Object operation makes the current registry object available for further operations.
Find_Data_Object_In_Repository	The Find_Data_Object_In_Repository operation locats a data object in the repository.
Find_Event_Subscribers	The Find_Event_Subscribers operation finds the subscribers to an event.
Find_Last_Registry_Object	The Find_Last_Registry_Object operation locates the last version of a registry object using its logical identifier.
Find_Metadata_Object_In_Repository	The Find_Metadata_Object_In_Repository operation locates a metadata object in the repository.
Find_Registry_Object	The Find_Registry_Object operation finds a registry object using its unique identifier.
Find_Registry_Object(Association)	The Find_Registry_Object(Association) operation finds a Association registry object.
Get_Association_From_User	The Get_Association_From_User operation gets an association (registry object) identifier from the user or system
Get_Classification_Node_From_User	The Get_Classification_Node_From_User operation gets a classification node (registry object) identifier from the user or system
Get_Data_Object_From_User	The Get_Data_Object_From_User operation gets the registry object's data object from the user.
Get_Metadata_Object_From_User	The Get_Metadata_Object_From_User operation gets the registry object's metadata from the user, including registry object identifers.
Get_Registry_Object_Id_From_User	The Get_Registry_Object_Unique_Id_From_User operation gets a registry object's unique identifer from the user.
Get_Registry_Object_Identifiers_From_User	The Get_Registry_Object_Identifiers_From_User operation gets a registry object's identifiers from the user. These could be the unique identifier, the logical identifier, or the logical identifier and version identifier.

Get_Registry_Object_Logical_Id_From_User	The Get_Registry_Object_Logical_Id_From_User operation gets a registry object's logical identifer from the user.
Get_Registry_Object_Status	The Get_Registry_Object_Status operation gets a registry object status from the user or system.
Get_Search_Parameters_From_User	The Get_Search_Parameters_From_User operation gets the registry catalog search parameters from the user. These parameters could include identifier, slots, classifications, and associations.
Metadata_Object_In_Repository	The Metadata_Object_In_Repository post condition indicates the registry object's metadata object has been successfully stored in the repository.
Notify_Subscribers	The Notify_Subscribers operation sends a notification message to the publisher and subscribers
PostCondition_Current_Registry_Object_Set_Available	The PostCondition_Current_Registry_Object_Set_Available post condition indicates that a set of registry objects has been located and their metadata objects and data objects are available.
Registry_Object_Not_Approved	The Registry_Object_Not_Approved precondition indicates that the registry object has been registered and has a catalog entry but is not approved..
Registry_Object_Not_Registered	The Registry_Object_Not_Registered precondition indicates that the registry object has not been registered and does not have a catalog entry.
Registry_Object_Registered	The Registry_Object_Registered precondition indicates that the registry object has been registered, has a catalog entry, and has been approved.
Registry_Object_Verrsion_Pre-exists	The Registry_Object_Verrsion_Pre-exists precondition indicates that a registry object with an identical logical_identifier has been previously registered.
Remove_Catalog_Entry	The Remove_Catalog_Entry operation removes the entry for a registry object from the registry catalog.
Remove_Data_Object_From_Repository	The Remove_Data_Object_From_Repository operation deletes the data object from the repository.
Remove_Metadata_Object_From_Re	The

pository	Remove_Metadata_Object_From_Repository operation deletes the metadata object from the repository.
Return_Status	The Return_Status operation returns the status to the user.
Search_Registry_Object	The Search_Registry_Object operation uses user provided search parameters to locate a registry object.
Set_Registry_Object_Status	The Set_Registry_Object_Status operation sets the status of a registry object.
Set_Registry_Object_Status_Approved	The Set_Registry_Object_Status_Approved operation sets the registry object status to Approved.
Set_Registry_Object_Status_Deprecated	The Set_Registry_Object_Status_Deprecated operation sets the registry object status to Deprecated
Set_Registry_Object_Status_Pending	The Operation:Set_Registry_Object_Status_Pending operation sets the registry object status to Pending.
Set_Registry_Object_Status_Validated	The Set_Registry_Object_Status_Validated operation sets the registry object status to Validated indicating that the metadata object has been validated.
Store_Data_Object_To_Repository	The Store_Data_Object_To_Repository operation places a data object into the repository
Store_Metadata_Object_To_Repository	The Store_Metadata_Object_To_Repository operation places a metadata object in the repository.
Update_Catalog_Entry	The Update_Catalog_Entry operation updates the registry catalog entry for a registry object.
Validate_Registry_Object_Metadata	The validate_Registry_Object_Metadata operation validates the metadata object.

6 FEDERATION

6.1 OVERVIEW

A federated registry/repository provides services for sharing content and metadata between cooperating registries in a federated environment; and allows cooperating registries to be federated together to appear and act as a single virtual registry/repository within the federated model. The benefits of which are evident in seamless information integration and sharing while preserving local autonomy over data (e.g., federated search seamlessly returns results from multiple stores).

The federated model includes features for federated query support, linking of content and metadata across registry boundaries, replication/synchronization of content and metadata among repositories, moving of content and metadata from one registry/repository to another, and event notifications. These capabilities enable the tying together of internal applications and the systems of the participating organizations in a federated architecture.

- Query – search registered content and metadata in any cooperating registry/repository (i.e., provide a seamless service across different registries in different domains).
- Linking – linking content and the associated metadata in any cooperating registry/repository (i.e., provide a seamless service across different registries in different domains).
- Replication/Synchronization – replication/synchronization of registered content and metadata between all cooperating registries (i.e., provide a seamless service across different registries in different domains).
- Relocation – relocation of registered content and metadata from one cooperating registry/repository to another (i.e., provide a seamless service across different registries in different domains).
- Notification – content-based event notification to registered client applications/systems to become aware of the latest information (i.e., provide a seamless service across different registries in different domains).

6.2 CONCEPT OF FEDERATION

A *federation* implies a loosely coupled system distributed across the Internet or an intranet, where the participants can join in and leave the federation without breaking the federation. It also implies that participants are autonomous independent entities that can function on their own when they are not a part of a federation. Each participant can support different schemas and their implementations can also be different. All participants do need to understand a common subset, which is represented by various federated models. That level of common understanding should suffice to create a federated architecture. An entity can participate in many federations at the same time and membership in a federation is not static. Each science organization typically maintains its own software systems (e.g., workflow, etc.) that cannot be dependent on systems of other organizations. These features make the federated architecture scalable and practical for science organizations.

A *federation* consists of more than one registry/repository that is self-governing but abides by a common set of rules to enable interoperability. The federation operates at a level where the participants within the federation are in agreement as to how to cooperate with respect to interoperability. Federation is expressed as a gradient of minimal interoperability to fully federated interoperability. Examples of various levels of federation; include:

- Minimally federated – entities share a minimal common subset (e.g., minimal rules and metadata; minimal access controls, etc.) where the federation operates as a loosely coupled system.
- Partially federated – entities share a larger common subset (e.g., half-measured set of rules and metadata; partial definition/enforcement over access controls, etc.) where the federation is represented by a semi-autonomous architecture.
- Fully federated – entities share a full common architecture where the federation operates using various federated models.

6.3 FEDERATED ARCHITECTURE

A federated architecture enables the individual cooperating organizations to function as a single federated system as illustrated in Figure 7. The federated architecture supports both large science organizations; as well as, small science organizations having limited resources.

The Federation class in the information model allows the creation of a Federation. A Federation is a registry/repository object and is registered and managed as any other registry/repository object.

The goal of a federated architecture is to create the appearance of a single “corporate” registry/repository while allowing individual organizations regional control over their individual realms. (“sub”-registries). One of the main requirements in achieving this goal is the ability to link and share information securely among sub-registries.

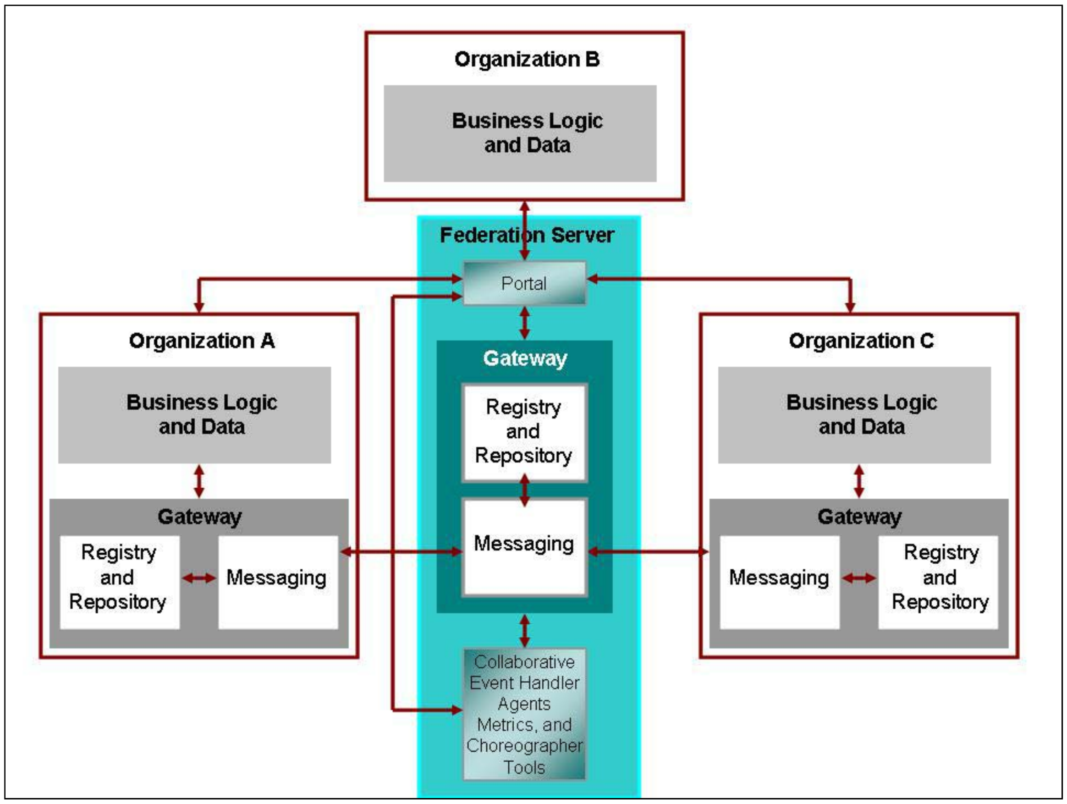


Figure 8: The Federated Reference Architecture

6.4 FEDERATED REGISTRY/REPOSITORY SERVICES

This section describes the capabilities and protocols that federated registries require in order to cooperate with each other for the following use cases. The use cases, capabilities, and protocols have been extracted from [7].

6.4.1 FEDERATED REGISTRY/REPOSITORY USE CASES

1. Inter-registry Object References - A submitter wishes to submit a `RegistryObject` such that the submitted object references a `RegistryObject` in another registry/repository.
2. Federated Queries - A client wishes to issue a single query against multiple registries and get back a single response that contains results based on all the data contained in all the registries. From the client's perspective it is issuing its query against a single logical registry/repository that has the union of all data within all the physical registries.
3. Local Caching of Data from Another Registry/Repository - A destination registry/repository wishes to cache some or all the data of another source registry/repository that is willing to share its data. The shared dataset is copied from the source registry/repository to the destination registry/repository and is visible to queries on the destination registry/repository even when the source

registry/repository is not available. Local caching of data may be desirable in order to improve performance and availability of accessing that object. An example might be where a `RegistryObject` in one registry/repository is associated with a `RegistryObject` in another registry/repository, and the first registry/repository caches the second `RegistryObject` locally.

4. Object Relocation - A Submitting Organization wishes to relocate its `RegistryObjects` and/or repository items from the registry/repository where it was submitted to another registry/repository.

6.4.2 REGISTRY/REPOSITORY FEDERATION

A registry/repository federation[4] is a group of registries that have voluntarily agreed to form a loosely coupled union. Such a federation may be based on common business interests and specialties that the registries may share. Registry/repository federations appear as a single logical registry/repository to registry clients.

Registry/repository federations are based on a peer-to-peer (P2P) model where all participating registries are equal. Each participating registry/repository is called a registry/repository peer. There is no distinction between the registry/repository operator that created a federation and those registry/repository operators that joined that Federation later. Any registry/repository operator MAY form a registry/repository federation at any time. When a federation is created it MUST have exactly one registry/repository peer which is the registry/repository operated by the registry/repository operator that created the federation.

Any registry/repository MAY choose to voluntarily join or leave a federation at any time.

The Federation information model is summarized here as follows:

- A Federation instance represents a registry/repository federation.
- A Registry/Repository instance represents a registry/repository that is a member of the Federation.
- An Association instance with associationType of HasFederationMember represents membership of the registry/repository in the federation. This Association links the Registry/Repository instance and the Federation instance.

[4] Significant amounts of material have been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [7] specification for this section.

6.4.3 QUERIES

A federation appears to registry/repository clients as a single unified logical registry/repository. A query, encoded into an instance of the class AdhocQueryRequest, is sent by a client to a federation member. The query may be local or federated as indicated by the boolean attribute "federated" in the instance of AdhocQueryRequest.

Local Queries - When the federated attribute of the query has the value of false then the query is a local query. A local AdhocQueryRequest is only processed by the registry/repository that receives the request. A local AdhocQueryRequest does not operate on data that belongs to other registries.

Federated Queries - When the federated attribute of AdhocQueryRequest has the value of true then the query is a federated query. A federation member MUST route a federated query received by it to all other federation member registries on a best attempt basis. When a registry/repository routes a federated query to other federation members it MUST set the federated attribute value to false and the federation attribute value to null to avoid infinite loops.

Membership in Multiple Federations - A registry/repository MAY be a member of multiple federations. In such cases if the federated attribute of AdhocQueryRequest has the value of true then the registry/repository MUST route the federated query to all federations that it is a member of.

6.4.4 FEDERATION LIFECYCLE MANAGEMENT PROTOCOLS

This section describes the various operations that manage the lifecycle of a federation and its membership. Federation lifecycle operations are done using the standard LifeCycleManager interface of the registry/repository in a stylized manner. Federation lifecycle operations are privileged operations. A registry/repository SHOULD Restrict Federation lifecycle operations to registry/repository User's that have the RegistryAdministrator role.

Joining a Federation - The following rules govern how a registry/repository joins a federation:

- Each registry/repository SHOULD have exactly one Registry/Repository instance within that registry/repository for which it is a home. The Registry/Repository instance is owned by the RegistryOperator and may be placed in the registry/repository using any operator specific means. The Registry/Repository instance SHOULD never change its home registry/repository.
- A registry/repository MAY request to join an existing federation by submitting an instance of an Extramural Association that associates the Federation instance as sourceObject, to its Registry/Repository instance as targetObject, using an

associationType of HasFederationMember. The home registry/repository for the Association and the Federation objects MUST be the same.

Creating a Federation - The following rules govern how a federation is created:

- A Federation is created by submitting a Federation instance to a registry/repository using SubmitObjectsRequest.
- The registry/repository where the Federation is submitted is referred to as the federation home.
- The federation home may or may not be a member of that Federation.
- A federation home MAY contain multiple Federation instances.

Leaving a Federation - The following rules govern how a registry/repository leaves a federation:

- A registry/repository MAY leave a federation at any time by removing its HasFederationMember Association instance that links it with the Federation instance. This is done using the standard RemoveObjectsRequest.

Dissolving a Federation - The following rules govern how a federation is dissolved:

- A federation is dissolved by sending a RemoveObjectsRequest to its home registry/repository and removing its Federation instance.
- The removal of a Federation instance is controlled by the same Access Control Policies that govern any `RegistryObject`.
- The removal of a Federation instance is controlled by the same lifecycle management rules that govern any `RegistryObject`. Typically, this means that a federation MUST NOT be dissolved while it has federation members. It MAY however be deprecated at any time. Once a Federation is deprecated no new members can join it.

7 EXTRINSIC OBJECTS

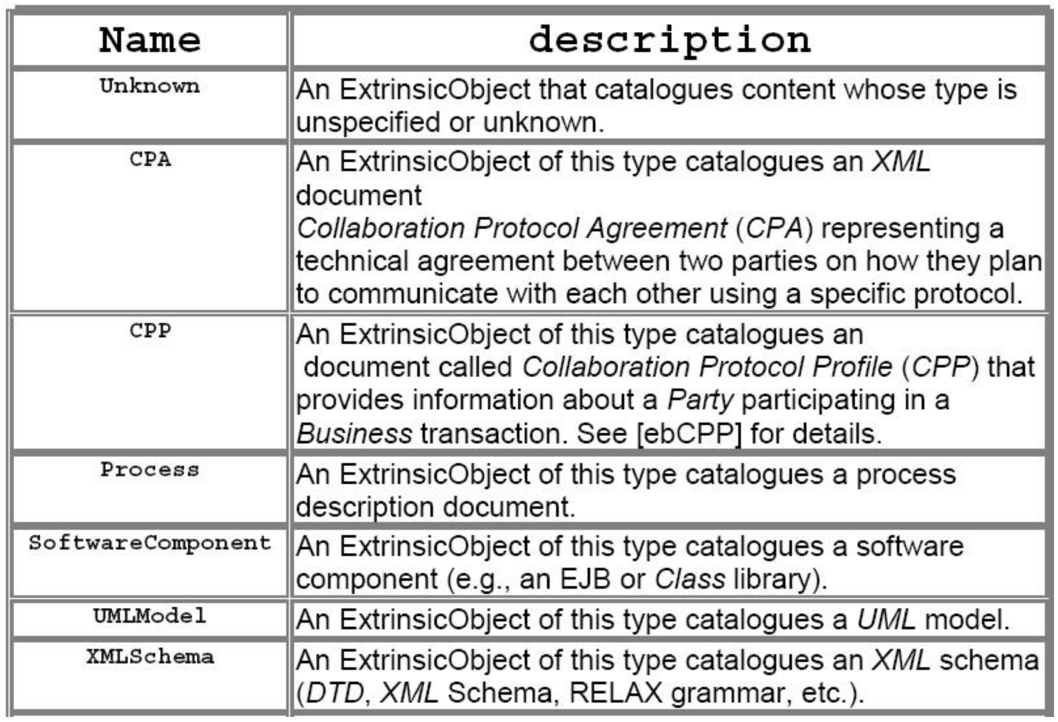
7.1 OVERVIEW

An *Extrinsic Object*[5] is a type of registry/repository object that catalogues content whose type is unspecified or unknown. Extrinsic Objects provide metadata that describes submitted content whose type is not intrinsically known to the registry/repository and therefore must be described by means of additional attributes. Since the registry/repository can contain arbitrary content without intrinsic knowledge about that content, Extrinsic Objects require special metadata attributes to provide some knowledge about the object (e.g., MIME type).

The super class for Extrinsic Object is `RegistryObject`. As a subclass it inherits all registered object attributes. Attributes defined specifically for the Extrinsic Object are “Is Opaque” and “mime Type”. The “Is Opaque” attribute determines whether the content catalogued by this Extrinsic Object is opaque to (not readable by) the Registry/Repository. In some situations, a Submitting Organization may submit content that is encrypted and not even readable by the registry/repository. The “mime Type” attribute provides information about the type of object since the object Type is user defined and not predefined in the registry/repository.

The following table lists pre-defined object types, for example schemas. Note that for an Extrinsic Object there are many types defined based on the type of repository item the Extrinsic Object catalogs. In addition there are object types defined for all leaf sub-classes of `RegistryObject`.

[5] Significant amounts of material have been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [7] specification for this section.



Name	description
Unknown	An ExtrinsicObject that catalogues content whose type is unspecified or unknown.
CPA	An ExtrinsicObject of this type catalogues an *XML* document *Collaboration Protocol Agreement* (*CPA*) representing a technical agreement between two parties on how they plan to communicate with each other using a specific protocol.
CPP	An ExtrinsicObject of this type catalogues an document called *Collaboration Protocol Profile* (*CPP*) that provides information about a *Party* participating in a *Business* transaction. See [ebCPP] for details.
Process	An ExtrinsicObject of this type catalogues a process description document.
SoftwareComponent	An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or *Class* library).
UMLModel	An ExtrinsicObject of this type catalogues a *UML* model.
XMLSchema	An ExtrinsicObject of this type catalogues an *XML* schema (*DTD*, *XML* Schema, RELAX grammar, etc.).

Table 2: Examples of Extrinsic Objects

7.1.1 EXTRINSIC OBJECT SUBCLASSES (CCSDS)

The following Extrinsic Object subclasses can be defined within the CCSDS and are provided as examples.

7.1.1.1 XML Schema

XML Schema is an extension of the `ExtrinsicObject` class. XML Schema is a W3C Recommendation and specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML documents. The XML Schema extension allows an organization to address XML Schema management functions, including registration, versioning, administer, store, and access using a CCSDS Registry/Repository.

An `ExtrinsicObject` has a boolean flag that indicates whether the content catalogued by the `ExtrinsicObject` is opaque to (not readable by) the registry/repository. See opaque attribute below.

The XML Schema extrinsic object is not opaque, and therefore allows the registry/repository to read and process the content. Content processing, such as decomposing the XML Schema and registering each component requires an augmentation to the registry/repository's generic capabilities.

Registering XML Schema components after decomposition will require that each component be defined as an extrinsic object. For example the XML Schema Element component will have to be defined.

The following required Event Types allow the tracking of XML Schemas.

- Approved - An Event that approves a `RegistryObject`.
- Created - An Event that created a `RegistryObject`.
- Deleted - An Event that deleted a `RegistryObject`.
- Deprecated - An Event that deprecated a `RegistryObject`.
- Downloaded - An Event that downloaded a `RegistryObject`.
- Relocated - An Event that relocated a `RegistryObject`.
- Undeprecated - An Event that undeprecated a `RegistryObject`.
- Updated - An Event that updated the state of a `RegistryObject`.
- Versioned - An Event that versioned a `RegistryObject`

In addition, each RegistryEntry instance must have a life cycle status indicator, assigned by the registry/repository. The following lists the pre-defined choices for `RegistryObject` status attribute.

- Submitted - Status of a `RegistryObject` that catalogues content that has been submitted to the Registry/Repository.
- Approved - Status of a `RegistryObject` that catalogues content that has been submitted to the Registry/Repository and has been subsequently approved.
- Deprecated - Status of a `RegistryObject` that catalogues content that has been submitted to the Registry/Repository and has been subsequently deprecated.
- Withdrawn - Status of a `RegistryObject` that catalogues content that has been withdrawn from the Registry/Repository.

Since `ExtrinsicObject` is a subclass of `RegistryObject`, the XML Schema class inherits the following `RegistryObject` attributes and is managed according to the registry/repository life-cycle protocols. In the following list the attributes are defined and restricted for use as a XML Schema.

isOpaque - This attribute determines whether the content catalogued by this `ExtrinsicObject` is opaque to (not readable by) the registry/repository. – For all XML Schemas, the value will be true. This implies that the registry/repository be able to read and process the content of the XML Schema.

mimeType - The mimeType provides information on the type of repository item catalogued by the `ExtrinsicObject` instance. – For all XML Schema the mimeType will be the XML Schema MimeType.

home - The home attribute, if present, MUST contain the base URL to the home registry/repository for the `RegistryObject` instance. No specific restriction.

Id - Each Identifiable instance MUST have a unique identifier that is used to refer to that object. No specific restriction.

Description - Each `RegistryObject` instance MAY have textual description in a human readable and user-friendly form. No specific restriction.

Lid - Each `RegistryObject` instance MUST have a lid (Logical Id) attribute. The lid is used to refer to a logical `RegistryObject` in a version independent manner. No specific restriction.

Name - Each `RegistryObject` instance MAY have a human readable name. The name does not need to be unique with respect to other `RegistryObject` instances. No specific restriction.

VersionInfo.Comment - Each VersionInfo instance MAY have a comment. This attribute defines the comment associated with the VersionInfo for a specific `RegistryObject` version. No specific restriction.

VersionInfoversion.Name - Each VersionInfo instance MUST have a versionName. This attribute defines the version name identifying the VersionInfo for a specific `RegistryObject` version. No specific restriction.

Slot.name - Each Slot instance MUST have a name. The name is the primary means for identifying a Slot instance within a `RegistryObject`. The Slot class is used to provide additional metadata for the `ExtrinsicObject`, beyond that defined for a standard `RegistryObject`. For the XML Schema extension, the Slot is used to indicate the query model attributes for finding the XML Schema.

Slot.slotType - Each Slot instance MAY have a slotType that allows different slots to be grouped together. The slotType attribute MAY also be used to indicate the data type or value domain for the slot value(s). See Slot.Name for XML Schema restrictions in general.

Slot.values - A Slot instance MUST have a Sequence of values. See Slot.Name for XML Schema restrictions in general.

ExternalIdentifier.value - Each ExternalIdentifier instance MUST have a value attribute that provides the identifier value for this ExternalIdentifier. No specific restriction. For an information system this could be a URI.

7.1.1.2 Content Information

The Content Information Object (CIO) is an extension of the `ExtrinsicObject` class. The CIO is defined within the OAIS [2] as "The set of information that is the original target of preservation." It consists of a content data object together with its representation data. The CIO extension allows science data information systems to address many of their archive ingest, administration, data management, archival storage, preservation, and access functional requirements using a CCSDS Registry/Repository.

For example, the archive tracking requirements can be met using Auditable Event Types. The following Event Types must be supported in a CCSDS Registry/Repository.

- Approved - An Event that approves a `RegistryObject`.
- Created - An Event that created a `RegistryObject`.
- Deleted - An Event that deleted a `RegistryObject`.
- Deprecated - An Event that deprecated a `RegistryObject`.
- Downloaded - An Event that downloaded a `RegistryObject`.
- Relocated - An Event that relocated a `RegistryObject`.
- Undeprecated - An Event that undeprecated a `RegistryObject`.
- Updated - An Event that updated the state of a `RegistryObject`.
- Versioned - An Event that versioned a `RegistryObject`

In addition, each RegistryEntry instance must have a life cycle status indicator, assigned by the registry/repository. The following lists the pre-defined choices for `RegistryObject` status attribute.

- Submitted - Status of a `RegistryObject` that catalogues content that has been submitted to the Registry.
- Approved - Status of a `RegistryObject` that catalogues content that has been submitted to the Registry and has been subsequently approved.
- Deprecated - Status of a `RegistryObject` that catalogues content that has been submitted to the Registry/Repository and has been subsequently deprecated.
- Withdrawn - Status of a `RegistryObject` that catalogues content that has been withdrawn from the Registry.

Since `ExtrinsicObject` is a subclass of `RegistryObject`, the CIO class inherits the following `RegistryObject` attributes and is managed according to the registry/repository life-cycle protocols. In the following list the attributes are defined and restricted for use as a CIO.

isOpaque - This attribute determines whether the content catalogued by this `ExtrinsicObject` is opaque to (not readable by) the registry/repository. – For all CIOs, the value will be false. This implies that the registry/repository will not care about the content of the CIO and the information system will be required to retrieve a CIO from the registry/repository for further processing.

mimeType - The mimeType provides information on the type of repository item catalogued by the `ExtrinsicObject` instance. – For all CIO's the mimeType will indicate parent information system and possible a CIO subclass. For example, within the

PDS, the mimeType will indicate that the CIO is a PDS data product and its subtype, such as an Image.

home - The home attribute, if present, MUST contain the base URL to the home registry/repository for the `RegistryObject` instance. No specific restriction.

Id - Each Identifiable instance MUST have a unique identifier that is used to refer to that object. No specific restriction.

Description - Each `RegistryObject` instance MAY have textual description in a human readable and user-friendly form. No specific restriction.

Lid - Each `RegistryObject` instance MUST have a lid (Logical Id) attribute. The lid is used to refer to a logical `RegistryObject` in a version independent manner. No specific restriction.

Name - Each `RegistryObject` instance MAY have a human readable name. The name does not need to be unique with respect to other `RegistryObject` instances. No specific restriction.

VersionInfo.Comment - Each VersionInfo instance MAY have a comment. This attribute defines the comment associated with the VersionInfo for a specific `RegistryObject` version. No specific restriction.

VersionInfoversion.Name - Each VersionInfo instance MUST have a versionName. This attribute defines the version name identifying the VersionInfo for a specific `RegistryObject` version. No specific restriction.

Slot.name - Each Slot instance MUST have a name. The name is the primary means for identifying a Slot instance within a `RegistryObject`. The Slot class is used to provide additional metadata for the `ExtrinsicObject`, beyond that defined for a standard `RegistryObject`. For the CIO extension, the Slot is used to indicate query model attributes for the information system. For example, within the PDS, the common data elements used for finding data products would be encoded into Slot, such as Time, Mission, Instrument, and Node. Discipline specific slot such as the imaging disciplines Latitude and Longitude could also be considered.

Slot.slotType - Each Slot instance MAY have a slotType that allows different slots to be grouped together. The slotType attribute MAY also be used to indicate the data type or value domain for the slot value(s). See Slot.Name for CIO restrictions in general. Slot.slotType would be used to differentiate between science disciplines specific queries such Imaging Latitude and Longitude and PPI regions.

Slot.values - A Slot instance MUST have a Sequence of values. See Slot.Name for CIO restrictions in general.

ExternalIdentifier.value - Each ExternalIdentifier instance MUST have a value attribute that provides the identifier value for this ExternalIdentifier. No specific restriction. For an information system this could be a URI.

7.1.1.3 Service

Since Services and Service Binding are first-class `RegistryObjects`, defined in the Registry Information Model, there is no need for a Registry Extension.

8 INTRINSIC OBJECTS

8.1 OVERVIEW

An *Intrisic Object*[6] is a type of registry/repository object that catalogues content whose type is specified and known. Intrinsic Objects have been defined in the registry schema and so their type is known to the registry/repository.

The super class for Intrinsic Object is `RegistryObject`. As a subclass it inherits all registered object attributes. Each Intrinsic object is then specifically defined.

The following table lists the defined intrinsic objects.

Intrinsic Object	Description
AdhocQuery	The AdhocQuery class is a container for an ad hoc query expressed in a query syntax that is supported
Association	Association instances are used to define many-to-many associations among RegistryObjects in the information model. An instance of the Association class represents an association between two RegistryObjects.
AuditableEvent	AuditableEvent instances provide a long-term record of events that effected a change in a RegistryObject. A RegistryObject is associated with an ordered Set of AuditableEvent instances that provide a complete audit trail for that RegistryObject. AuditableEvents are usually a result of a client-initiated request. AuditableEvent instances are generated by the Registry Service to log such Events. Often such events effect a change in the life cycle of a RegistryObject. For example a client request could Create, Update, Deprecate or Delete a RegistryObject. An AuditableEvent is typically created when a request creates or alters the content or ownership of a RegistryObject. Read-only requests typically do not generate an AuditableEvent.
Classification	A Classification instance classifies a RegistryObject instance by referencing a node defined within a particular ClassificationScheme. An internal Classification will always reference the node directly, by its id, while an external Classification will reference the node indirectly by specifying a representation of its value that is unique within the external classification scheme. The attributes

[6] Significant amounts of material have been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [7] specification for this section.

	for the Classification class are intended to allow for representation of both internal and external classifications in order to minimize the need for a submission or a query to distinguish between internal and external classifications.
ClassificationNode	ClassificationNode instances are used to define tree structures where each node in the tree is a ClassificationNode. Such ClassificationScheme trees are constructed with ClassificationNode instances under a ClassificationScheme instance, and are used to define Classification schemes or ontologies.
ClassificationScheme	A ClassificationScheme instance describes a taxonomy. The taxonomy hierarchy may be defined internally to the registry by instances of ClassificationNode, or it may be defined externally to the Registry, in which case the structure and values of the taxonomy elements are not known to the Registry. In the first case the classification scheme is said to be internal and in the second case the classification scheme is said to be external.
ExternalIdentifier	ExternalIdentifier instances provide the additional identifier information to RegistryObject such as DUNS number, Social Security Number, or an alias name of the organization. The attribute identificationScheme is used to reference the identification scheme (e.g., "DUNS", "Social Security #"), and the attribute value contains the actual information (e.g., the DUNS number, the social security number). Each RegistryObject MAY contain 0 or more ExternalIdentifier instances.
ExternalLink	ExternalLinks use URIs to associate content in the registry with content that MAY reside outside the registry. For example, an organization submitting an XML Schema could use an ExternalLink to associate the XML Schema with the organization's home page.
ExtrinsicObject	The ExtrinsicObject class is the primary metadata class for a RepositoryItem.
Federation	Federation instances are used to represent a registry federation.
Notification	The Notification class represents a Notification from the registry regarding an event that matches a Subscription. A registry may uses a Notification instance to notify a client of an event that matches a Subscription they have registered. This is a push model of notification. A client may also pull events from the registry using the AdhocQuery protocol defined by [ebRS].
Organization	Organization instances provide information on organizations such as a Submitting Organization. Each Organization instance MAY have a reference to a parent

	Organization.
Person	Person instances represent persons or humans.
Registry	Registry instances are used to represent a single physical OASIS ebXML Registry.
RegistryPackage	RegistryPackage instances allow for grouping of logically related RegistryObject instances even if individual member objects belong to different Submitting Organizations.
Service	Service instances describe services, such as web services.
ServiceBinding	ServiceBinding instances are RegistryObjects that represent technical information on a specific way to access a Service instance. An example is where a ServiceBinding is defined for each protocol that may be used to access the service.
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 SpecificationLink instances that describe how to access the service using a technical specification such as a WSDL document or a CORBA IDL document.
Subscription	Subscription instances are RegistryObjects that define a User's interest in certain types of AuditableEvents. A User MAY create a subscription with a registry if he or she wishes to receive notification for a specific type of event.

Table 3: Defined Intrinsic Objects

9 LIFECYCLE MANAGEMENT

9.1 OVERVIEW

This section[7] defines the protocols supported by the Lifecycle Management service interface of the registry. The Lifecycle Management protocols provide the functionality required by `RegistryClients` to manage the lifecycle of `RegistryObjects` and `RepositoryItems` within the registry. These lifecycle protocols have been extracted from [7].

9.2 UPDATE OBJECTS PROTOCOL

The UpdateObjectsRequest protocol allows a Registry Client to update one or more existing `RegistryObjects` and/or repository items in the registry.

UpdateObjectsRequest - The UpdateObjectsRequest is used by a client to update `RegistryObjects` and/or repository items that already exist within the registry.

RegistryObjectList: This parameter specifies a collection of `RegistryObject` instances that are being updated within the registry.

9.3 APPROVE OBJECTS PROTOCOL

The Approve Objects protocol allows a client to approve one or more previously submitted `RegistryObject` objects using the LifeCycleManager service interface.

ApproveObjectsRequest - The ApproveObjectsRequest is used by a client to approve one or more existing `RegistryObject` instances in the registry.

Parameters:

- AdhocQuery: This parameter specifies a query. A registry MUST approve all objects that match the specified query in addition to any other objects identified by other parameters.
- ObjectRefList: This parameter specifies a collection of references to existing `RegistryObject` instances in the registry. A registry MUST

[7] Significant amounts of material have been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [7] and the ebXML Registry Information Model (ebXML RIM) [3] specifications for this section.

approve all objects that are referenced by this parameter in addition to any other objects identified by other parameters.

9.4 DEPRECATE OBJECTS PROTOCOL

The Deprecate Object protocol allows a client to deprecate one or more previously submitted `RegistryObject` instances using the LifeCycleManager service interface. Once a `RegistryObject` is deprecated, no new references (e.g. new Associations, Classifications and ExternalLinks) to that object can be submitted. However, existing references to a deprecated object continue to function normally.

DeprecateObjectsRequest - The DeprecateObjectsRequest is used by a client to deprecate one or more existing `RegistryObject` instances in the registry.

Parameters:

- AdhocQuery: This parameter specifies a query. A registry MUST deprecate all objects that match the specified query in addition to any other objects identified by other parameters.
- ObjectRefList: This parameter specifies a collection of references to existing `RegistryObject` instances in the registry. A registry MUST deprecate all objects that are referenced by this parameter in addition to any other objects identified by other parameters.

9.5 UNDEPRECATE OBJECTS PROTOCOL

The Undeprecate Objects protocol of the LifeCycleManager service interface allows a client to undo the deprecation of one or more previously deprecated `RegistryObject` instances. When a `RegistryObject` is undeprecated, it goes back to the Submitted status and new references (e.g. new Associations, Classifications and ExternalLinks) to that object can now again be submitted.

UndeprecateObjectsRequest - The UndeprecateObjectsRequest is used by a client to undeprecate one or more existing `RegistryObject` instances in the registry. The registry MUST silently ignore any attempts to undeprecate a `RegistryObject` that is not deprecated.

Parameters:

- AdhocQuery: This parameter specifies a query. A registry MUST undeprecate all objects that match the specified query in addition to any other objects identified by other parameters.

- ObjectRefList: This parameter specifies a collection of references to existing `RegistryObject` instances in the registry. A registry MUST undeprecate all objects that are referenced by this parameter in addition to any other objects identified by other parameters.

9.6 REMOVE OBJECTS PROTOCOL

The Remove Objects protocol allows a client to remove one or more `RegistryObject` instances and/or repository items using the LifeCycleManager service interface.

RemoveObjectsRequest - The RemoveObjectsRequest is used by a client to remove one or more existing `RegistryObject` and/or repository items from the registry.

Parameters:

- deletionScope: This parameter indicates the scope of impact of the
- RemoveObjectsRequest. The value of the deletionScope attribute MUST be a reference to a ClassificationNode within the canonical DeletionScopeType ClassificationScheme.
- AdhocQuery: This parameter specifies a query. A registry MUST remove all objects that match the specified query in addition to any other objects identified by other parameters.
- ObjectRefList: This parameter specifies a collection of references to existing `RegistryObject` instances in the registry. A registry MUST remove all objects that are referenced by this parameter in addition to any other objects identified by other parameters.

9.7 REGISTRY MANAGED VERSION CONTROL

This section describes the version control features of the registry.

Version Controlled Resources - All repository items in a registry are implicitly version-controlled resources. No explicit action is required to make them a version-controlled resource.

Versioning and Object Identification - Each version of a `RegistryObject` is a unique object and as such has its own unique value for its id attribute as defined by the information model.

Logical ID - All versions of a `RegistryObject` are logically the same object and are referred to as the logical `RegistryObject`. A logical `RegistryObject` is a tree structure where nodes are specific versions of the `RegistryObject`.

A specific version of a logical `RegistryObject` is referred to as a `RegistryObject` instance. A `RegistryObject` instance MUST have a Logical ID (LID) to identify its membership in a particular logical `RegistryObject`. Note that this is in contrast with the id attribute that MUST be unique for each version of the same logical `RegistryObject`. A client may refer to the logical `RegistryObject` in a version independent manner using its LID.

Version Identification - A registry supports independent versioning of both `RegistryObject` metadata as well as repository item content. It is therefore necessary to keep distinct version information for a `RegistryObject` instance and its repository item if it happens to be an `ExtrinsicObject` instance.

Version Identification for a `RegistryObject` - A `RegistryObject` MUST have a versionInfo attribute whose type is the VersionInfo class defined by information model. The versionInfo attributes identifies the version information for that `RegistryObject` instance. A registry MUST not allow two versions of the same `RegistryObject` to have the same versionInfo.versionName attribute value.

Versioning of `ExtrinsicObject` and Repository Items - An `ExtrinsicObject` and its associated repository item may be updated independently and therefore versioned independently.

Version Creation - The registry manages creation of new version of a `RegistryObject` or a repository item automatically. A registry that supports versioning MUST implicitly create a new version for a repository item if the repository item is updated via a SubmitObjectsRequest or UpdateObjectsRequest. In such cases it MUST also create a new version of its `ExtrinsicObject`.

10 REFERENCE IMPLEMENTATION

The Planetary Data System (PDS) has undertaken an effort to overhaul the PDS data architecture (e.g., data model, data structures, data dictionary, etc.) and deploy a software system (online data services, distributed data catalog, etc.) that fully embraces the PDS federation as an integrated system while leveraging modern information technology. The effort has been dubbed “PDS 2010”.

A core component of this new system is the Registry Service that will provide functionality for tracking, auditing, locating, and maintaining artifacts within the system. These artifacts can range from data files and label files, schemas, dictionary definitions for objects and elements, services, etc. The design of this service attempts to follow the reference model captured in this document.

10.1 SERVICE DESCRIPTION

The Registry Service provides a generalized track and locate function for registered artifacts within the system. Services and individual actors within the system interact with the registry to inform the service about new managed artifacts or to lookup/update basic information about existing registered artifacts. The registry will maintain three types of registrations:

Metadata Entry

This type of entry will simply capture metadata describing a non-digital object within the system. This type of entry is akin to descriptions captured for missions, instruments, data sets, targets, people, etc.

Digital Object Entry

This type of entry tracks back to a physical set of bits. This would be items such as science data products consisting of a label and data files. It also includes any item of interest (e.g., documents, schemas, etc.).

Relationship Entry

This type of entry will serve as a means to tie registered artifacts together. Such support is necessary for example to correlate collections to a set of products contained within. These relationships may span registries and thus the need for coordination amongst registries exists. Example science data product relationships include associations with an investigation product, an instrument product and a target product.

Although the service follows the reference model captured in this document, it does deviate from the model in four areas:

Repository Support

Each Node within the PDS federation manages their science archive (e.g., the digital objects) utilizing their own mechanism. Because of this, the need for a common or generalized repository capability has been downplayed. As a result, the design of the Registry Service focuses on registration of metadata for digital objects that are managed in an external repository.

Search Support

The design of the PDS 2010 system includes a Search Service where search indices are built from harvested metadata supporting the different science disciplines and search applications. Although the Registry Service facilitates search and retrieval, the focus is on facilitating the harvest of metadata from the registry in order to build these search indices.

Security Support

The design of the PDS 2010 system includes a Security Service where user accounts and groups are maintained for controlling access to the Registry Service. In addition, this service works with the Application Server hosting the Registry Service to restrict access and require authentication to specific URLs (i.e., any interface that modifies the contents of the registry).

Subscription Support

The design of the PDS 2010 system includes a Subscription Service that manages user subscriptions and the mechanisms for notification. The Registry Service tracks auditable events and provides an interface for retrieving these events in order to facilitate subscription.

The Registry Service offers a single implementation of registry capabilities for use by the other services and applications within a system. This service is tailor-able depending on the type of registry and types of artifacts to be registered with a given instance.

10.2 USE CASES / REQUIREMENTS

The use cases for the Registry Service follow the general use cases defined in Section 3.2 very closely. In fact they are virtually identical except for the Store and Retrieve use cases found in this document that pertain to repository-related capabilities. In addition, the requirements for the registry also map to the actions associated with those general use cases. Because of this very close mapping to the use cases and associated actions found in Section 3.2, the use cases and requirements for the Registry Service will not be detailed in this document.

10.3 ARCHITECTURE

The architecture of the Registry Service focuses mainly on the interfaces offered by the service.

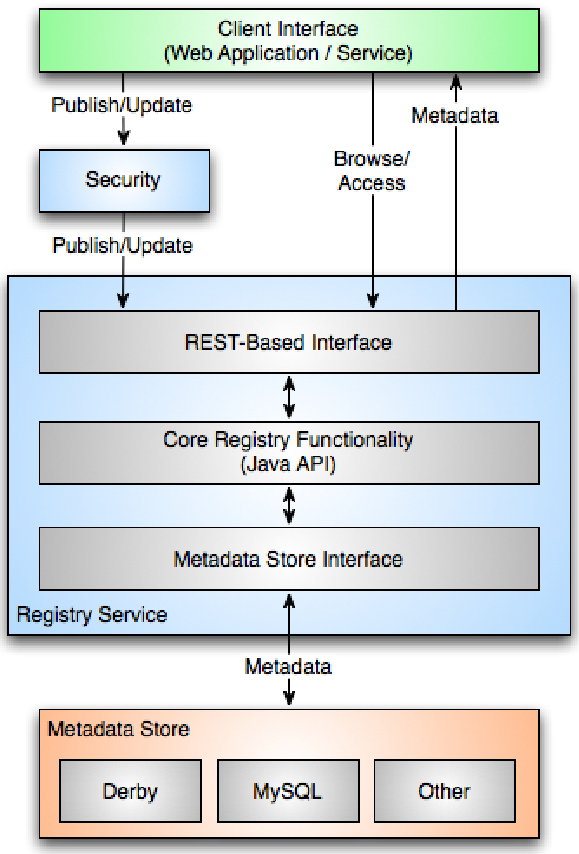


Figure 9: Registry Service Architecture

The interfaces depicted in the figure above are described in the following sections.

10.3.1 EXTERNAL INTERFACE

The Registry Service offers a REST-based external interface that is accessible via the Hypertext Transfer Protocol (HTTP). A REST-based interface exhibits the following characteristics:

- A URL assigned to every resource
- Formulate URLs in a predictable manner
- Use HTTP methods for actions on a resource (GET, POST and DELETE)
- Leverage HTTP protocol headers and response codes where applicable

The goals for the interface are as follows:

- Keep the service simple and refrain from adding too much functionality
- Allow messaging in the form of XML or JavaScript Object Notation (JSON)
- Allow for extensibility as new artifact types are defined

In addition, each interface should adhere to the following:

- Be self documenting
- Have a defined standard response including passed parameters
- Provide a schema for the defined response
- Provide a command-line method of execution

Any interface that modifies the contents of the registry will incorporate security. This means that any interface specified below as an HTTP POST will first require interaction with a Security service. Integration with the Security service is accomplished through the Application Server and does not require any specific coding within the Registry Service. The only change to these interfaces will be in terms of a required HTTP header or cookie being set that will provide the means to verify the validity of the request. These requests should require secure HTTP (HTTPS).

The following are some examples detailing the functionality of the REST-based interface using HTTP methods. This interface delegates all functions involving a product:

http://pds.nasa.gov/services/registry/products/
- GET: Retrieves a paged list of products from the registry.
- POST: Publishes a product to the registry.

This interface acts on a specific product (lid stands for logical identifier):

http://pds.nasa.gov/services/registry/products/{lid}/{version}/
- GET: Retrieves the product from the registry.
- POST: Updates the product in the registry.
- DELETE: Removes the product from the registry.

10.3.2 INTERNAL INTERFACES

The Registry Service defines two internal interfaces. The first interface exposes the core functionality of the service via a Java API. This API is utilized by the REST-based interface detailed in the previous section. The following table maps this Java API to the façade API defined in Section 5.3:

Façade API	**Registry Service API**
Publish_Registry_Object	RegistryService.publishRegistryObject()

Façade API	**Registry Service API**
Publish_Registry_Object_No_Data	RegistryService.publishRegistryObject()
Update_Registry_Object	RegistryService.updateRegistryObject()
Approve_Registry_Object	RegistryService.changeRegistryObjectStatus()
Deprecate_Registry_Object	
Undeprecate_Registry_Object	
Delete_Registry_Object	RegistryService.deleteRegistryObject()
Validate_Registry_Object	RegistryService.publishRegistryObject()
Catalog_Registry_Object	RegistryService.publishRegistryObject()
Version_Registry_Object	RegistryService.versionProduct()
Store_Data_Object	Not Supported
Notify_Subscribers	Not Supported. Facilitated by an external Subscription Service via a call to: RegistryService.getAuditableEvents()
Find_Registry_Object	RegistryService.getProduct()
Discover_Registry_Object	RegistryService.getProducts()
Retrieve_Data_Object	Not Supported
Retrieve_Metadata_Object	RegistryService.getRegistryObject()

Façade API	**Registry Service API**
Set_Registry_Object_Status Set_Registry_Object_Status_Pending Set_Registry_Object_Status_Validated Set_Registry_Object_Status_Approved Set_Registry_Object_Status_Deprecated	RegistryService.changeRegistryObjectStatus()
Find_All_Associated_Object	RegistryService.getAssociation()
Associate_From_Object Associate_To_Object	RegistryService.publishRegistryObject()
Deassociate_From_Object Deassociate_To_Object	RegistryService.deleteRegistryObject()
Classify_Object	Not Yet Supported
Publish_Association	RegistryService.publishRegistryObject()
Publish_Classification	RegistryService.publishRegistryObject()
Publish_Federation	Not Yet Supported
Publish_AuditableEvent	RegistryService.publishRegistryObject()
Publish_Notification	Not Supported
Publish_Subscriber	Not Supported
Join_Federation	Not Yet Supported

Table 4: Registry Service API Mapping

The second internal interface involves communication with the underlying metadata store. This interface will follow a generic design with the intent of supporting multiple backend implementations for the metadata store. The layered design for the backend implementation allows for technology refresh and multiple deployment scenarios. The metadata store interface will support the data model detailed in Section 4.2.1.2.

10.4 DEPLOYMENT

The implementation platform for the Registry Service is the Java 2 Platform Standard Edition 6.0. Implementation of the REST-based interface utilizes Jersey, which is a reference implementation of the Java API for RESTful Web Services (JAX-RS) framework. The Graphical User Interface (GUI) that interfaces with the Registry Service was also developed using Java and the Google Web Toolkit. In addition, development utilizes publicly available libraries for interface development, message handling and file system access. The service and the GUI are packaged as Web Application Archives (WARs), which require an Application Server (e.g., Apache Tomcat) installed on the target machine to host the applications. The following diagram depicts this deployment scenario:

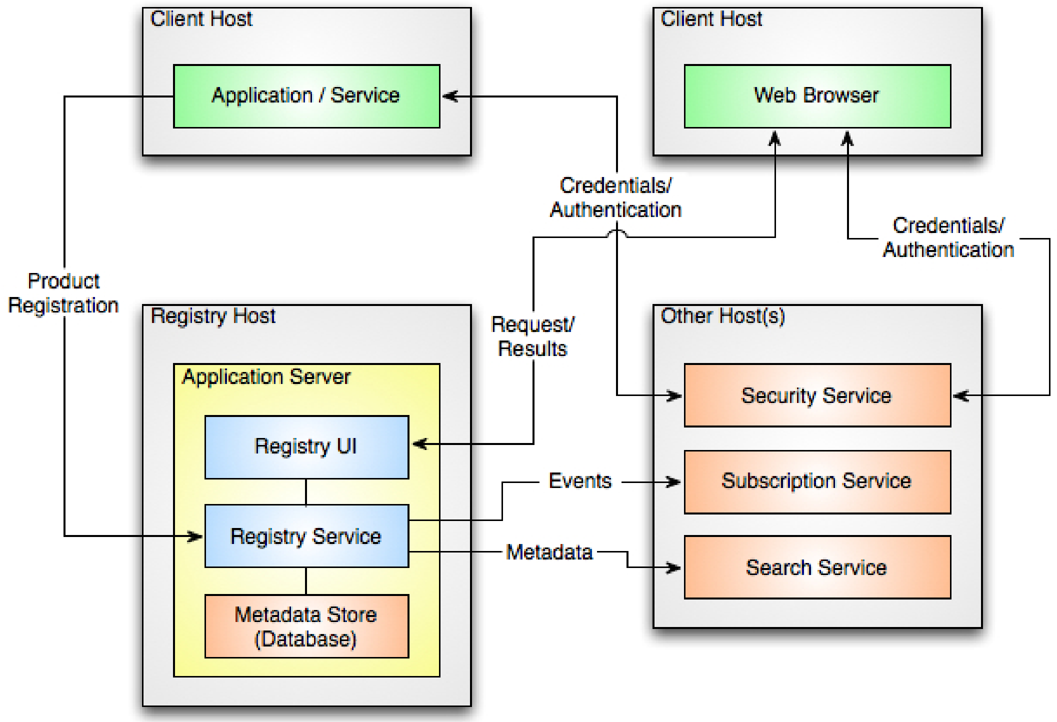


Figure 10: Registry Service Deployment

Communication with the Registry Service is accomplished using REST-based API over the Hypertext Transfer Protocol (HTTP).

ANNEX Sections

Annex 1 Electronic Business using XML Registry (ebXML Registry) [8]

1.5 Electronic Business using XML Registry (ebXML Registry)

The ebXML Registry specification was created as part of the 18-month ebXML initiative that ended in May 2001. Sponsored by the United Nations Centre for Trade Facilitation and 345 Electronic Business (UN/CEFACT) and OASIS, ebXML is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. ebXML provides companies with a standard method to exchange business messages, conduct trading relationships, communicate data in common terms, and define and register business processes. An ebXML registry provides a mechanism by which XML artifacts can be stored, maintained, and automatically discovered, thereby increasing efficiency in XML-related development efforts [7]. The OASIS/ebXML Registry Technical Committee was created in May 2001 to build on the ebXML initiative efforts. The current ebXML Registry standard is ebXML Registry v3.0. It was ratified by OASIS in May 2005, superseding the previous standard, ebXML Registry v2.0. The ebXML Registry specification is actually comprised of two specifications—ebXML Registry Information Model (RIM) and ebXML Registry Services (RS). These specifications are referred to collectively here as the "ebXML Registry specification."

Although ebXML Registry may be used as a general-purpose registry, the stated goal in the specification is to facilitate ebXML-based B2B partnerships and transactions. Unfortunately, very few SOA infrastructure products or tools provide any support for ebXML-compliant registry services [7]. The tool vendors have instead favored supporting UDDI standard for managing metadata associated with services assets.

1.5.1 ebXML Registry Information Model and APIs & Protocols

Unlike UDDI whose primary focus is business information, the main focus of the ebXML Registry Information Model (RIM) is more general to encompass XML and non-XML artifacts. Therefore, the ebXML RIM is more abstract in nature than that of UDDI. The ebXML RIM consists of two "core" data structures, or classes:

- RegistryObject
- RegistryEntry

A `RegistryObject` provides metadata for a stored RepositoryItem (the term used to refer to that actual object that is stored) – such as name, object type, identifier, description, etc. A `RegistryObject` can represent many different types of `RepositoryItems`, from XML schemas, to classification schemes, to Web service

[8] Extracted from "Registry & Repository Standards Introduction,", Jeff Estefan, Division 31 Chief Technologist, Jet Propulsion Laboratory.

definitions [7]. In contrast, a RegistryEntry is used to represent “catalog-type” metadata about `RepositoryItems`—that is, metadata about the current state of a RepositoryItem in the registry (e.g. version, status, stability). Consequently, the metadata associated with a RegistryEntry is (in general) more “fluid” than that associated with a `RegistryObject`. The RegistryEntry class inherits from the `RegistryObject` class. An ebXML Registry service must support Simple Object Access Protocol (SOAP) and HTTP protocol bindings. The ebXML Registry v3.0 specification defines the following APIs [7]:

- Query Management (to browse and search the registry)
- Lifecycle Management (to publish information to the registry, manage content in the repository, and manage versions)
- Event Notification (to subscribe to changes in the registry or repository)
- Content Management (to validate and catalog content in the repository)

The ebXML Registry specification defines two levels of compliance, Registry Full and Registry Lite. A Registry Fullcompliant product must implement all features defined in the specification. The Registry Lite compliance level defines version control, event notification, and content management as optional features.

1.5.2 ebXML Registry withing ebXML Technical Architecture

The ebXML Registry is a central component of the ebXML Technical Architecture, as it serves as a storage facility and discovery mechanism for the various artifacts that are necessary for engaging in electronic business using the ebXML framework. This is illustrated in Error! Reference source not found.

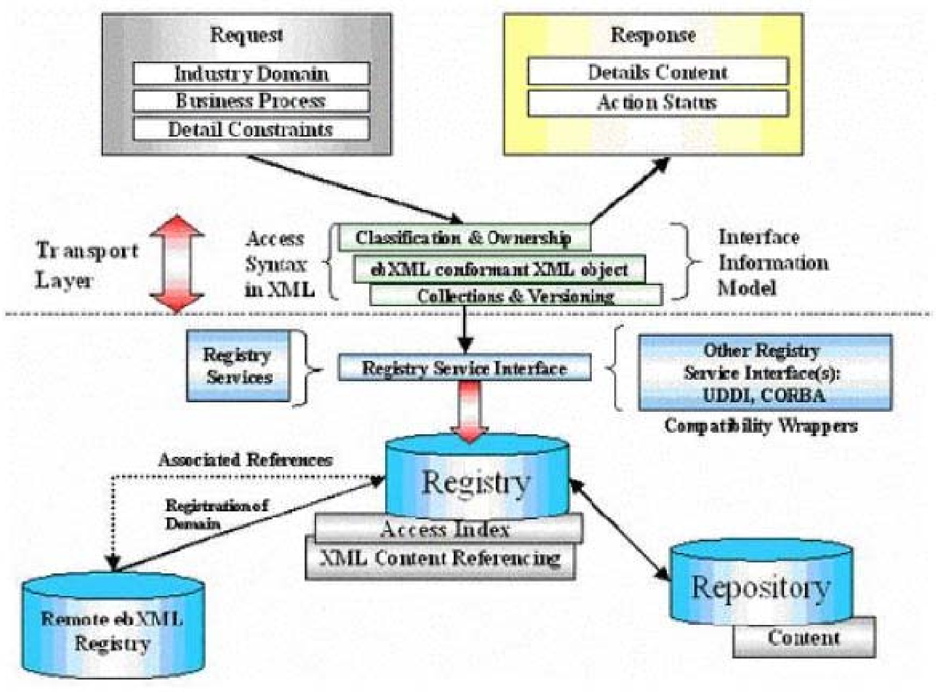


Figure 1 - ebXML Technical Architecture

An ebXML registry interacts with both a local repository and a remote ebXML registry. Requests are sent to the registry and responses are received from the registry through a Registry Service Interface. The Registry Service Interface may interact with other Registry Service Interfaces, such as UDDI, and open interface standards such as Common Object Request Broker Architecture (CORBA).

Annex 2 – XML Schema Registry Use Cases

10.5 XML SCHEMA REGISTRY/REPOSITORY

This section provides use cases and actors for an XML Schema registry/repository. An XML Schema registry/repository allows for the registration, discovery and management of XML Schemas. The XML Schema registry/repository inherits several of functions of the general registry/repository described earlier.

10.5.1 ACTORS

XML Schema Provider - An XML schema provider is a system or person that provides a schema to be submitted into the registry/repository. This actor is equivalent to the Publisher defined in Section 3.1.

XML Schema Consumer - An XML schema consumer is a system or person that discovers and receives descriptions and schemas from the registry/repository. This actor is equivalent to the Artifact Consumer defined in Section 3.1.

10.5.2 USE CASES

The following are use case defined for a XML Schema Registry.

10.5.2.1 XML Schema Registration

Description: An XML Schema Provider registers an XML schema through a registry/repository service.

Actors/Users: XML Schema Provider

Scenarios

1. An XML Schema Provider registers an XML schema file through a registry/repository service using standardized metadata.

2. The registry/repository service associates the metadata to the XML schema in the catalog

3. The registry/repository service allows classification of the registered XML schema in the catalog based on classification schemes in the registry (including namespace)

4. The registry/repository service assigns a version identifier for the XML schema in the catalog

5. The registry/repository service stores the XML schema in the repository

6. The registry/repository service returns a unique identifier for accessing the schema

10.5.2.2 XML Schema Search

Description: An XML Schema Consumer searches for existing XML schemas through a registry/repository service.

Actors/Users: XML Schema Consumer

Scenarios

1. An XML Schema Consumer searches the XML schema registry/repository based on a set of criteria.

2. The registry/repository service returns metadata results describing registered XML schemas.

10.5.2.3 XML Schema Access

Description: An XML Schema Consumer accesses a schema based on a unique identifier

Actors/Users: XML Schema Consumer

Scenarios

1. An XML Schema Consumer retrieves an XML schema file through a registry/repository service using a unique identifier for the schema and version (e.g., URL)

2. The registry/repository service retrieves the XML schema from the repository and returns it to the consumer

10.5.2.4 XML Schema Validation

Description: A XML Schema Consumer validates an XML document against a schema in the registry/repository

Actors/Users: XML Schema Consumer

Scenarios

1. An XML Schema Consumer provides an XML document to the XML Schema registry/repository service requesting that it be validated against a registered XML schema based on the unique identifier

2. The registry/repository service performs the validation and returns the results to the consumer

10.5.2.5 XML Schema Update

Description: A XML Schema Provider updates the metadata for a XML schema in the registry/repository

Actors: XML Schema Provider

Scenarios

1. An XML Schema Provider provides updates to the metadata for a registered XML schema

2. The registry/repository service updates the metadata for the XML schema in the catalog

10.5.2.6 XML Schema New Version

Description: An XML Schema Provider registers a new version of a schema

Actors: XML Schema Provider

Scenarios

1. An XML Schema Provider registers a new version of an XML schema file through a registry/repository service using standardized metadata.

2. The registry/repository service associates the metadata to the XML schema in the catalog

3. The registry/repository service allows classification of the registered XML schema in the catalog based on classification schemes in the registry/repository (including namespace)

4. The registry/repository service increments a version identifier for the XML schema in the catalog

5. The registry/repository service stores the XML schema in the repository

6. The registry/repository service returns a unique identifier for accessing the schema

10.5.2.7 XML Schema Removal

Description: An XML Schema Provider requests the XML schema be removed from the registry/repository

Actors: XML Schema Provider

Scenarios

1. An XML Schema Provider requests the XML Schema registry/repository to remove the schema based on a unique identifier

2. The registry/repository service removes the schema from the repository

3. The registry/repository service removes the description from the catalog