Consultative Committee for Space Data Systems

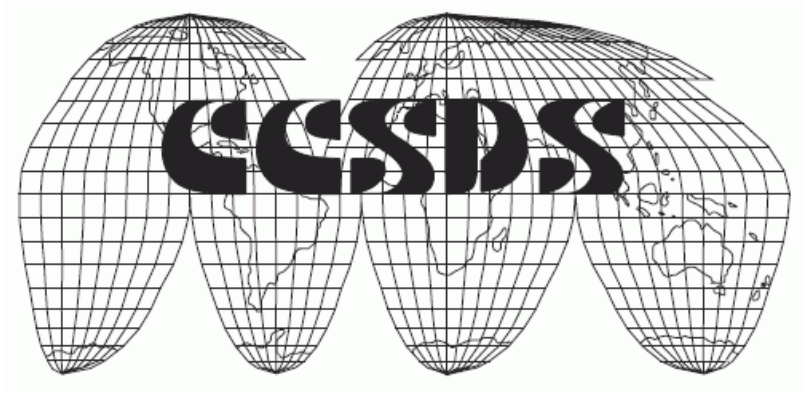
RECOMMENDATION FOR SPACE DATA SYSTEM STANDARDS

Registry and Repository Reference Model

CCSDS [number]

DRAFT WHITE BOOK

June 25, 2009


CCSDS

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.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- British National Space Centre (BNSC)/United Kingdom.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency(JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- Russian Space Agency (RSA)/Russian Federation.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- Centro Tecnico Aeroespacial (CTA)/Brazil.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Communications Research Laboratory (CRL)/Japan.
- Danish Space Research Institute (DSRI)/Denmark.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Federal Science Policy Office (FSPO)/Belgium.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space and Astronautical Science (ISAS)/Japan.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- National Oceanic & Atmospheric Administration (NOAA)/USA.
- National Space Program Office (NSPO)/Taipei.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.

– United States Geological Survey (USGS)/USA.

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

CONTENTS

Section Page

Table of Figure

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 material has been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [13] and the ebXML Reference Information Model (ebXML RIM) [5] 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.

Comment [JAE1]: You should point out that there are different types of metadata as you have articulated in the CCSDS RASIM and cite that reference by page number.

Comment [JAE2]: IMPORTANT!: We need to state right up front when we talk about a "registry/repository," we're talking about a registry that may have a repository backend. We're not talking about describing a spec for the generic term of repository such as a file system or database.

A simple graphic would be helpful here.

LET'S TALK ABOUT THIS BEFORE GOING FORWARD.

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 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 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 6 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 7 presents a standard Java 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 8 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.

Annexes 1-3

- Annex 1 provides sets of use cases for specific systems
- Annex 2 provides the mapping for the JAXR API's to the use cases of Chapter 4 and to the CCSDS XML/Schema tool APIs.
- Annex 3 provides the JAXR API APIs

Comment [JAE3]: Include a very brief definition of what an *extrinsic object* is here.

Comment [JAE4]: What is this? An interface definition?

Comment [JAE5]: These look like JAXR things. If it is JAXR, there should be some brief statement that that API set is being referenced here with forward reference to Annex 3.

1.4 DEFINITIONS

1.4.1 ACRONYMS AND ABBREVIATIONS

AIC	Archival Information Collection
AIP	Archival Information Package
AIU	Archival Information Unit
ASCII	American Standard Code for Information Interchange
CCSDS	Consultative Committee for Space Data Systems
CD-ROM	Compact Disk - Read Only Memory
CORBA	Common Object Request Broker Architecture
CRC	Cyclical Redundancy Check
DIME	Direct Internet Message Encapsulation
DIP	Dissemination Information Package
ebXML	Electronic Business using eXtensible Markup Language
FITS	Flexible Image Transfer System
GIF	Graphics Interchange Format
ISBN	International Standard Book Number
ISO	International Organization for Standardization
METS	Metadata Encoding and Transmission Standard
MIME	Multipurpose Internet Mail Extensions
OAIS	Open Archival Information System
OWL	Web Ontology Language
PDI	Preservation Description Information
PDS	Planetary Data System
RDF	Resource Description Format
SFDU	Standard Formatted Data Unit
SIP	Submission Information Package
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery & Integration
UML	Unified Modeling Language
UNICODE	Universal Code
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Wed Consortium
WWW	Worldwide Web
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

Content Data Object: The Data Object, which together with 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.

Context Information: The information that documents the relationships of the Content Information to its environment. This includes why the Content Information was created and how it relates to other Content Information objects.

Content Objects: The data and/or metadata objects, and any Content Units, logically within a given Content Unit.

Content Unit: A structure that contains pointers to data objects and associated metadata objects, and possibly other Content Units.

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 Dictionary: A formal repository of terms used to describe data.

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

Data Object Section: Contains a number of data Object elements

Description Data Unit: A Content Unit where all the content objects are metadata objects.

Descriptive Information: The set of information, consisting primarily of Package Descriptions, which is provided to Data Management to support the finding, ordering, and retrieving of OAIS information holdings by Consumers.

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.

Digital Object: An object composed of a set of bit sequences.

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: Data about other data.

Physical Object: An object (such as a moon rock, bio-specimen, microscope slide) with physically observable properties that represent information that is considered suitable for being adequately documented for preservation, distribution, and independent usage.

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.

Structure Information: The information that imparts meaning about how other information is organized. For example, it maps bit streams to common computer types such as characters, numbers, and pixels and aggregations of those types such as character strings and arrays.

Submission Information Package (SIP): An Information Package that is delivered by the Producer to the OAIS for use in the construction of one or more AIPs.

Comment [HJS(6]: Many of the terms in this list are no longer used in the document and should probably be deleted.

1.5 REFERENCES

Comment [JAE7]: References need to be cleaned-up. Some of the are JPL internal docs and do not recommend reference to internal docs that Intl community cannot gain access. Such references if required should be added as footnotes.

[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.

[4] Open Gis Project Document - Registry Service - Version: 0.3

Comment [JAE8]: What is this?

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

[6] 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.

[7] Use Cases for the DSMS Information Services Architecture Registry (Draft), M. Demore Editor, Jet Propulsion Laboratory, Oct 2004.

[8] Registry Pilot Task, Costin Radulescu, 9/17/2007, Presentation.

[9] ESA Use Cases

[10] SM&C Use Cases

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

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

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

Comment [JAE9]: This is an internal JPL document and not available in the public domain. It is also dated. Recommend use of the more recent DISA RegRep OpsCon document.

Comment [JAE10]: Incomplete reference citations.

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 and Repository recommendation. A Registry addresses the following essential functional requirements:

Comment [JAE11]: Sounds like the functions of a repository. Really need to make the distinction somewhere between a registry and a repository (i.e., virtual system of record vs. datastore or both).

- 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 AND REPOSITORY

Figure 1 illustrates a registry and repository in the context of a generic layered system environment [8]. The registry/repository foundation in the framework includes features for query support, linking of content and metadata, replication and synchronization of content and metadata, 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. The use of existing standards, such as Java™ API for XML Registries (JAXR), helps illustrate the maturity of the registry/repository concept. JAXR in particular supports registry operations over a diverse set of registries and defines a unified information model for describing registry contents.

JAXR is used in the figure simply as an example of an existing Application Programming Interface (API) standard. Finally, access control and data management modules, tools, and a governance model bridge the functionality gap to support the enterprise applications.

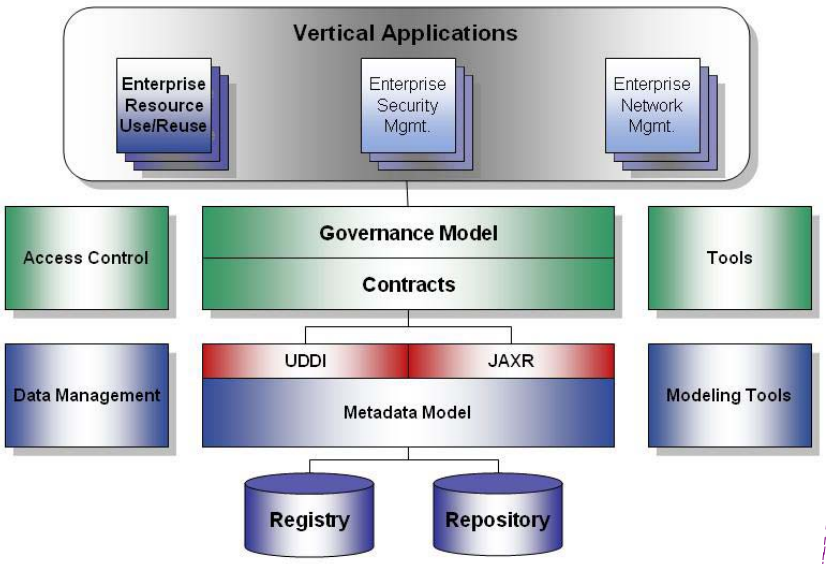


Figure 1 Environment View of a Registry and Repository

Comment [JAE12]: Shows UDDI and JAXR but not ebXML RIM. Need to add latter to support text in Sect 2.2 that follows. Should show JAXR sitting above UDDI and ebXML RIM where JAXR is the API and UDDI and ebXML RIM are the registry/repository standards.

Comment [JAE13]: Here, you show Registry and Repository as separate stores. I think we need to be clear that this spec covers interfacing with the registry that may have a repository store as a backend.

2.2 FUNCTIONAL VIEWS OF A REGISTRY AND 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.

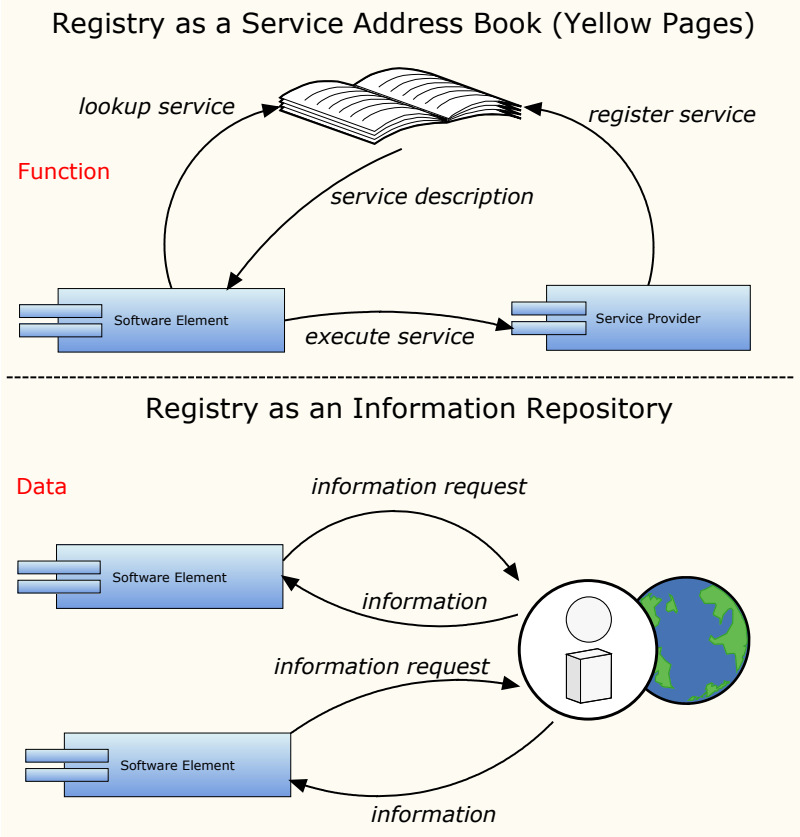


Figure 2: Two Conceptual Views of a Registry/Repository

2.3 GENERAL FEATURES OF A REGISTRY AND 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.

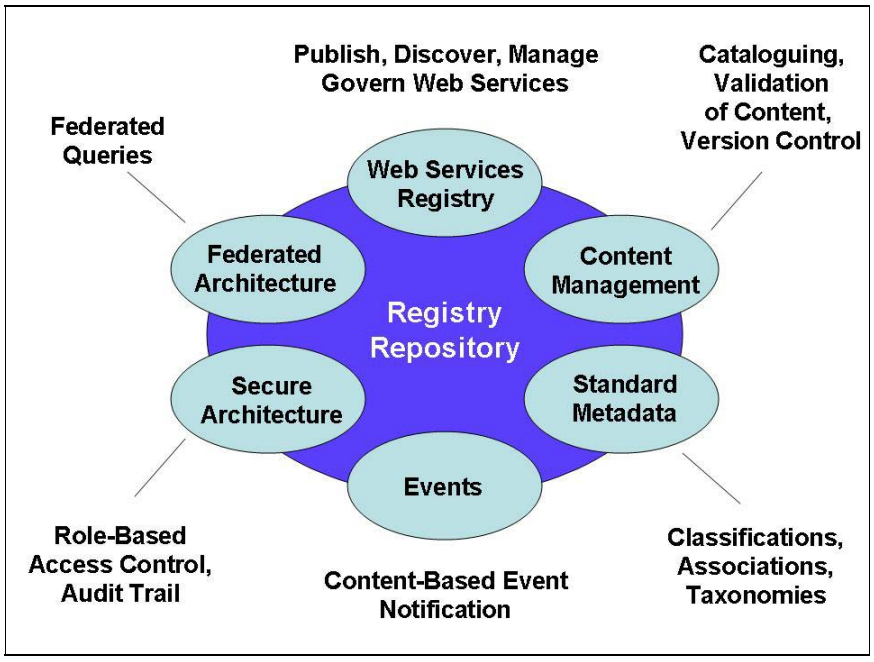


Figure 3: Features of a Federated Registry/Repository

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.

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 artefact 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

[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].

registry/repository functions which include changes to the catalog and/or repository.

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. In addition, use cases for specific objects are addressed in subsequent sections of this document.

Figure 4 illustrates a generalized view of the interactions between actors and registry/repository services [8], the interface available for performing registry/repository functions.

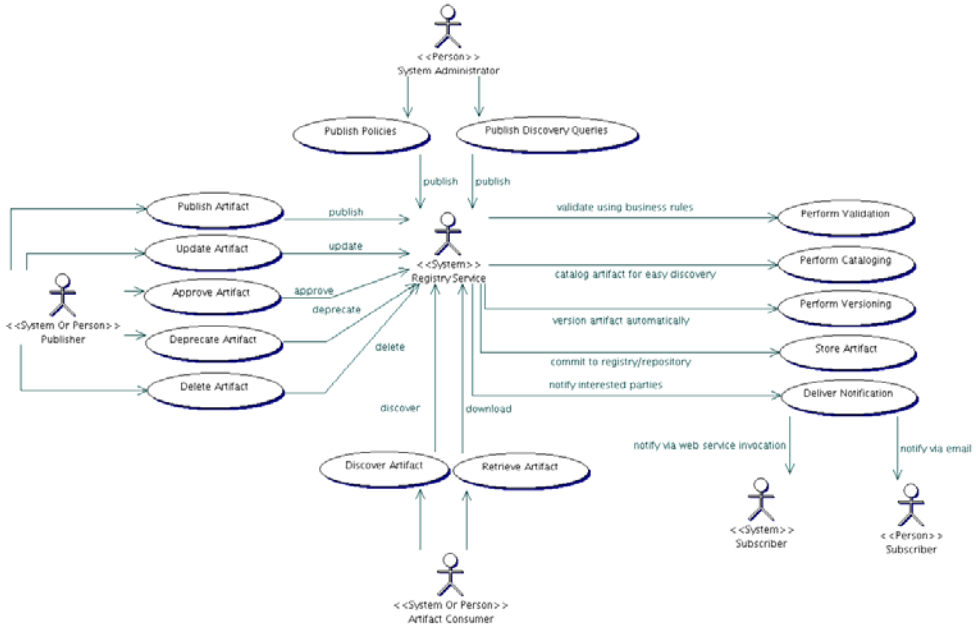


Figure 4 General Registry/Repository Use Cases

In Annex 2, the general use cases in Figure 4 are mapped to the JAXR APIs.

3.2.1 PUBLISH

Description

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

Actors: Registry Publisher

Actions:

1. Registry/Repository Publisher publishes a new artifact in the registry which includes descriptive metadata about the artifact
2. Registry/Repository Service validates the metadata
3. Registry/Repository Service assigns a version identifier for the artifact
4. Registry/Repository Service updates the catalog with the metadata
5. Registry/Repository Service stores the artifact in the repository
6. Registry/Repository Service returns an identifier for the published artifact and the version
7. Registry/Repository Service sends notification to the subscribers regarding the published artifact.

3.2.2 UPDATE

Description

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

Actors: Registry Publisher

Actions:

1. User 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
3. Registry/Repository Service updates the metadata for the artifact in the catalog
4. Notification is sent to the subscribers of the update

3.2.3 APPROVE

Description

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

Actors: System Administrator

Actions:

1. User queries the registry/repository service for newly published artifacts which are not already approved
2. User updates the metadata for an artifact to indicate whether it is approved or rejected
3. Notification is sent to the publisher and subscribers

3.2.4 DEPRECATE

Description

This use case describes the actions necessary to deprecate an artifact in the registry/repository.

Actors: System Administrator, Registry Publisher

Actions:

1. Publisher updates the registry/repository with a new version of an artifact, if applicable

2. System administrator updates the registry/repository to indicate that a specific artifact and version is deprecated.

3. Subscribers are notified of the deprecated artifact.

3.2.5 DELETE

Description

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

Actors: Registry Publisher

Actions:

1. User 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

3.2.6 VALIDATE

Description

This use case describes the actions necessary to validate the metadata associated with an artifact.

Actors: System Administrator

Actions:

1. User publishes a new artifact which includes descriptive metadata about the artifact

2. Registry/Repository Service validates the metadata

Comment [HJS(14]: From Jean-Christophe Malapert - I read the Registry&Repository document and I have one comment about the use case "validate".

Only the metadata coming from the service seems to be validated and I think we should validate the service itself. In my past experience in the virtual observatory, I used to query the virtual observatory registry fo find out all services matching a set of criterions. However, most of the service did not work or the answer was wrong (the contain of the metadata was fine but the quality of metadata was wrong for positional search). At the end, VO registries were not queried directly by these sofwares. The softwares were queried a internal registry cleaned from wrong services.

In the use case "validate", do you consider checking each registered service in a period of time to validate the metadata quality ? Do you consider to check whether a server is still available ?

3.2.7 CATALOG

Description

This use case describes the actions necessary to catalog a new artifact.

Actors: System Administrator

Actions:

1. User publishes a new artifact which includes descriptive metadata about the artifact
2. Registry/Repository Service updates the catalog with the metadata

3.2.8 VERSION

Description

This use case describes the actions necessary to version a new artifact.

Actors: System Administrator

Actions:

1. User publishes a new artifact
2. Registry/Repository Service assigns a version identifier to the artifact

3.2.9 STORE

Description

This use case describes the actions necessary to store the artifact.

Actors: System Administrator

Actions:

1. User publishes a new or updated artifact
2. Registry/Repository Service updates to the repository with the artifact

3.2.10 NOTIFY

Description

This use case describes the actions necessary to subscribe to registry/repository events.

Actors: Registry Consumer

Actions:

1. User creates a subscription for the registered event
2. Registry/Repository Service notifies the user when the event has occurred

3.2.11 DISCOVER

Description

This use case describes the actions necessary to discover registered artifacts

Actors: Registry Consumer

Actions:

1. User enters search criteria
2. Registry searches the catalog and returns metadata describing registered artifacts that meet the search criteria.

3.2.12 RETRIEVE

Description

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

Actors: Registry Consumer

Actions:

1. User enters an identifier for the artifact

2. Registry 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 include 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 actions described in 4.2. At present, these are specifically service and XML schema registry/repository use cases.

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

Service consumer – A service consumer is a system or person that discovers and receives descriptions of services in the registry/repository

3.4.1.2 Service Registration Use Cases

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

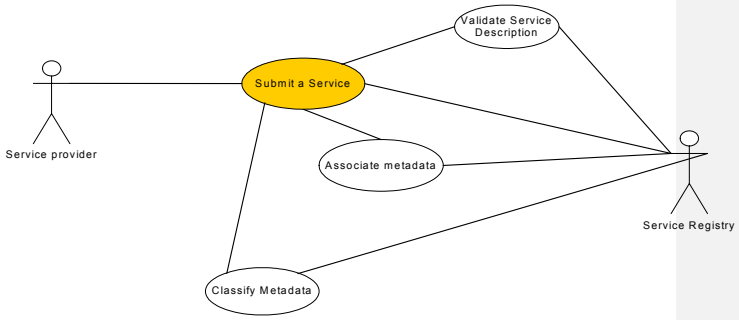
Actors: Service Provider

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 Use Cases

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

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 Use Cases

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

Actors: Service Provider, Service Consumer

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

3.4.2 XML SCHEMA REGISTRY/REPOSITORY

This section provides use cases 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.

3.4.2.1 Actors

XML schema provider - A XML schema provider is a system or person that provides a schema to be submitted into the registry/repository

XML schema consumer – A XML schema consumer is a system or person that discovers and receives descriptions and schemas from the registry/repository.

3.4.2.2 XML Schema Registration Use Cases

Description: A Data Provider registers an XML schema through a registry/repository service.

Actors/Users: XML Schema Provider

Scenarios

1. A 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

3.4.2.3 XML Schema Search Use Cases

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

Actors/Users: XML Schema Consumer

Scenarios

1. A 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.

3.4.2.4 XML Schema Access Use Cases

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

Actors/Users: XML Schema Consumer

Scenarios

1. A XML Schema Provider 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

3.4.2.5 XML Schema Validation Use Cases

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

Actors/Users: XML Schema Consumer

Scenarios

1. A XML schema registry/repository consumer provides an XML document to the XML 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

3.4.2.6 XML Schema Update Use Cases

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

Actors: XML Schema Provider

Scenarios

1. A 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

3.4.2.7 XML Schema New Version Use Cases

Description: A XML schema registry/repository provider registers a new version of a schema

Actors: XML Schema Provider

Scenarios

1. A XML Schema Provider registers a new version of a 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

3.4.2.8 XML Schema Removal Use Cases

Description: A XML schema registry/repository provider requests the XML schema be removed from the registry/repository

Actors: XML Schema Provider

Scenarios

1. A 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

4 INFORMATION MODEL

The CCSDS Registry Information Model Specification document [12] formally defines the proposed information model. At an abstract level, this information model, based on the ebXML Reference Information Model (RIM) [5], is considered to be the model that typically results from registry/repository designs that support general registry/repository use cases and functional requirements. 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.

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 Publish, Discover, and Manage 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. The current specification leverages the work done in the OASIS [OAS] and the ISO 11179 [ISO] Registry/Repository models.

4.1.1 RESPONSE TO USE CASES

The following table provides the information model response to the registry/repository use cases. For each use case, the associated actions are used to derive functional requirements for registry/repository functions. From the functional requirements the information model classes and the instantiated objects needed by the registry/repository to perform the associated function are derived. This response matrix verifies that the ebXML information model components can be derived from the general use cases in section 2.

Use Cases	**Derived Services/Tools and Functional Requirements** -- Provide services and tools that implement system functions. Note that services and tools perform the functions that implement the functional requirements by acting on the associated models.	**Derived Requirements affecting Data Architecture** -- define standard models that support System Services.	**Suggested Clas**
Publish Description This use case describes the actions necessary for a user to publish an artifact in the registry Actors: Registry Publisher, Registry Service	1. User publishes a new artifact in the registry which includes descriptive metadata about the artifact 2. Registry service validates the metadata 3. Registry service assigns a version identifier for the artifact 4. Registry service updates the catalog with the metadata 5. Registry service stores the artifact in the repository 6. Registry service returns an identifier for the published artifact and the version 7. Notification is sent to the subscribers regarding the published artifact	A registry shall have models for the following concepts: **user, artifact, registry, descriptive metadata, version identifier, catalog, repository, identifier, version, notification, subscriber, service**	**Registry, RegistryObject, ClassificationScheme, Identifiable, ContentInformation, Person, Organization, Service, Notification, Subscription, User, VersionInfo, Repository.**
Update Description This use case describes the actions necessary to update an artifact in the registry Actors: Registry Publisher, Registry Service	1. User requests that an artifact be updated based on an identifier and version for the artifact 2. Registry service replaces the artifact in the repository 3. Registry service updates the metadata for the artifact in the catalog 4. Notification is sent to the subscribers of the update		

Comment [JAE15]: Comments regarding dropping the registry/repository as an actor apply here as well. Is there some way this table can be cleaned up for readability purposes?

Approve Description This use case describes the actions necessary to approve an artifact in the registry Actors: System Administrator, Registry Service	1. User queries the registry service for newly published artifacts which are not already approved 2. User updates the metadata for an artifact to indicate whether it is approved or rejected 3. Notification is sent to the publisher and subscribers	A registry shall have models for the following concepts: **query, search constrains, status, publisher**	**QueryExpression, StatusType, publisher**
Deprecate Description This use case describes the actions necessary to deprecate an artifact in the registry Actors: System Administrator, Publisher, Registry Service	1. Publisher updates the registry with a new version of an artifact, if applicable 2. System administrator updates the registry to indicate that a specific artifact and version is deprecated. 3. Subscribers are notified of the deprecated artifact.	A registry shall have models for the following concepts: **system adminitrator,**	**system adminitrator**
Delete Description This use case describes the actions necessary to delete an artifact in the registry Actors: Registry Publisher, Registry Service	1. User requests that an artifact be deleted based on an identifier for the artifact 2. Registry service deletes the artifact from the repository 3. Registry service removes the artifact from the catalog		
Validate Description This use case describes the actions necessary to validate the metadata associated with an artifact Actors: Registry Service	1. User publishes a new artifact which includes descriptive metadata about the artifact 2. Registry service validates the metadata		

Catalog Description This use case describes the actions necessary to catalog a new artifact Actors: Registry Service	1. User publishes a new artifact which includes descriptive metadata about the artifact 2. Registry service updates the catalog with the metadata		
Version Description This use case describes the actions necessary to version a new artifact Actors: Registry Service	1. User publishes a new artifact 2. Registry service assigns a version identifier to the artifact		
Store Description This use case describes the actions necessary to store the artifact Actors: Registry Service	1. User publishes a new or updated artifact 2. Registry service updates to the repository with the artifact		
Notify Description This use case describes the actions necessary to subscribe to registry events Actors: Registry Consumer, Registry Service	1. User creates a subscription for the registered event 2. Registry service notifies the user when the event has occurred	A registry shall have models for the following concepts: **registered event**	**EventType**
Discover Description This use case describes the actions necessary to discover registered artifacts Actors: Registry Consumer, Registry Service	1. User enters search criteria 2. Registry searches the catalog and returns metadata describing registered artifacts that meet the search criteria.	A registry shall have models for the following concepts: **search criteria**	
Retrieve Description This use case describes the actions necessary to retrieve a registered artifacts Actors: Registry Consumer, Registry Service	1. User enters an identifier for the artifact 2. Registry retrieves the artifact from the repository and returns the artifact in its original form	A registry shall have models for the following concepts: **package**	**RegistryPackage, ExtrinsicObject**

Table 1 – Information Model Response to Use Cases

4.1.2 VIEWS OF THE REGISTRY/REPOSITORY MODEL

The CCSDS Registry/Repository Information Model Specification document [12] provides a formal data engineering definition of the registry/repository

information model captured from the ebXML Registry Information Model (RIM) specification. [5] In the following two sections high level conceptual and logical views of the formation model are provided.

4.1.2.1 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 2 - CCSDS Registry/Repository Components and Functions

Comment [JAE16]: Need a cleaner table.

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

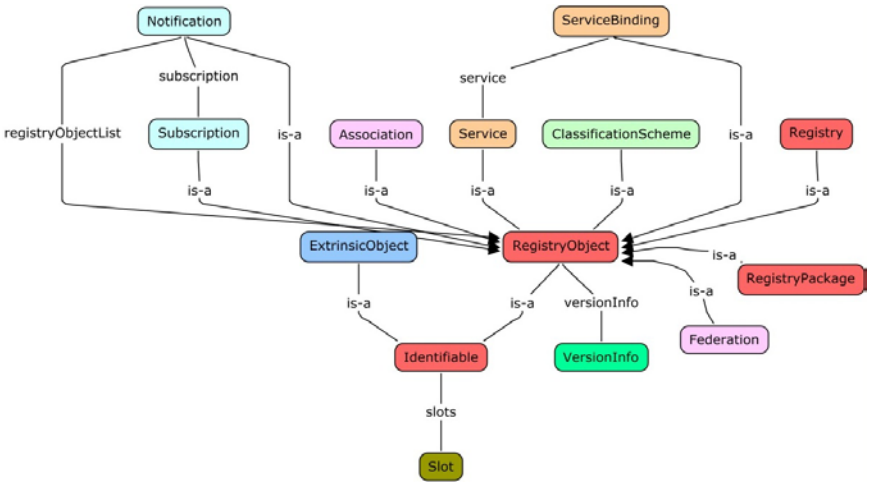


Figure 5 - CCSDS Registry/Repository Conceptual Map – Key Classes

4.1.2.2 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.

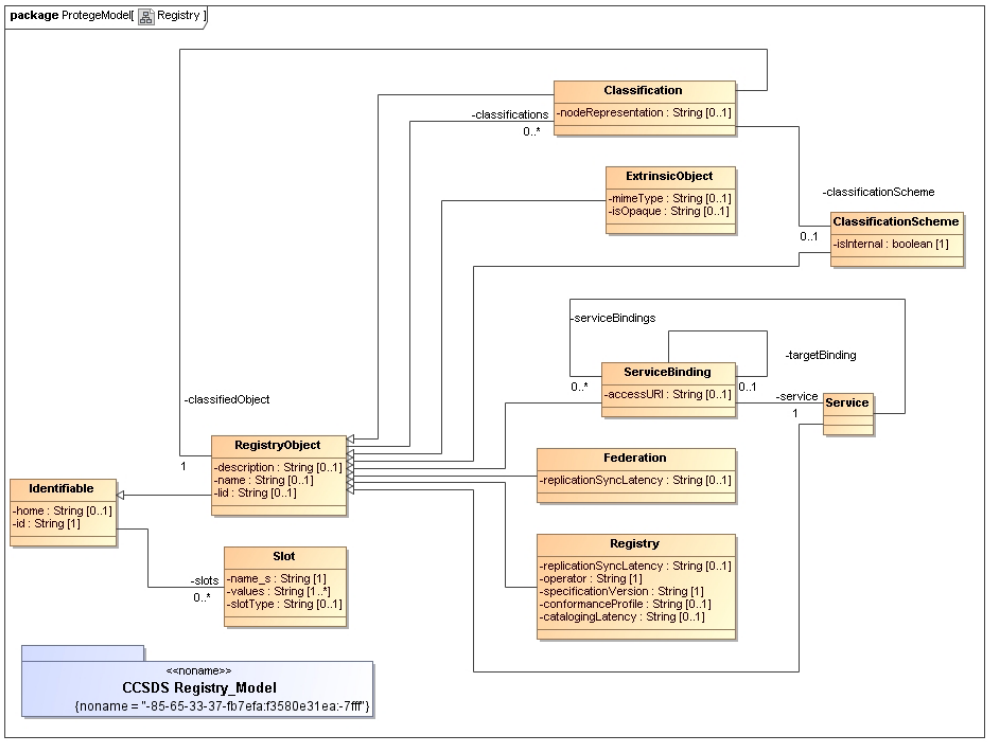


Figure 6 – Key Registry/Repository Class Definitions

Comment [JAE17]: This UML diagram shows the concept of ExtrinsicObject but it is not defined and discussed until Section 6. Therefore, recommend swapping Sections 5 and 6.

5 FEDERATION

5.1 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).

5.2 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:

1. 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.
2. 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 an semi-autonomous architecture.
3. Fully federated – entities share a full common architecture where the federation operates using various federated models.

5.3 FEDERATED ARCHITECTURE

A federated architecture enables the individual cooperating organizations to function as a single federated system. 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.

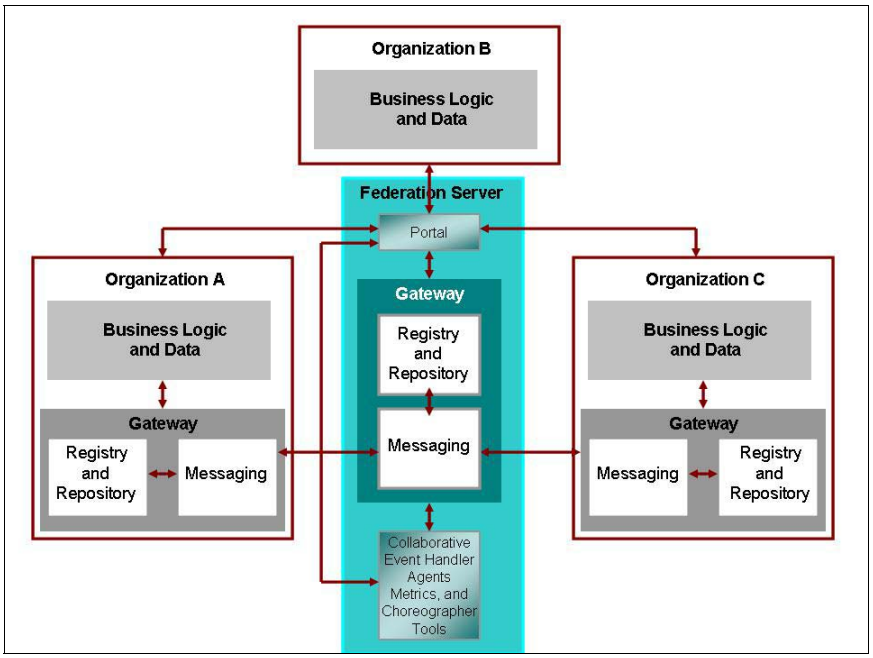


Figure 7. The Federated Reference Architecture

Comment [JAE18]: Is there an external reference for this graphic or is it something we made up?

5.4 FEDERATED REGISTRY/REPOSITORY SERVICES

This section describes the capabilities and protocols that federated registries to cooperate with each other for the following use cases. The use cases, capabilities, and protocols have been extracted from [13].

5.4.1 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.

Comment [JAE19]: Recommend making all code like fragments or class and interface names use Courier New font. This is but one example. Need to make the change throughout the document.

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.

5.4.2 REGISTRY/REPOSITORY FEDERATION

A registry/repository federation[3] 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.

Comment [JAE20]: Use of MUST or MAY or SHOULD are IETC RFC 2119 requirements that you typically see in an OASIS and other specs that drive conformance specs. Is that also the convention used by CCSDS?

[3] Significant amounts of material have been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [13] specification for this section.

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.

5.4.3 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.

5.4.4 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 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.

6 EXTRINSIC OBJECTS

6.1 OVERVIEW

An *Extrinsic Object*[4] 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.

Comment [JAE21]: Recommend making all code like fragments or class and interface names use Courier New font. This is but one example. Need to make the change throughout the document.

[4] Significant amounts of material have been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [13] 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 3 - Examples of Extrinsic Objects

6.1.1 EXTRINSIC OBJECT SUBCLASSES (CCSDS)

The following Extrinsic Object subclasses are defined within the CCSDS.

6.1.1.1 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.

A 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.

- Created - An Event that created a RegistryObject.
- Deleted - An Event that deleted a RegistryObject.
- Deprecated - An Event that deprecated 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 which 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 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 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 query model attributes for finding 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 a information system this could be a URI.

6.1.1.2 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.

- Created - An Event that created a RegistryObject.
- Deleted - An Event that deleted a RegistryObject.
- Deprecated - An Event that deprecated 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 which 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 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 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 a information system this could be a URI.

6.1.1.3 Service

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

7 APPLICATION PROGRAMMING INTERFACE (API)

7.1 OVERVIEW

This specification defines a façade[5] 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 x?. Figure 10 illustrates the façade API that hides the diverse registry APIs.

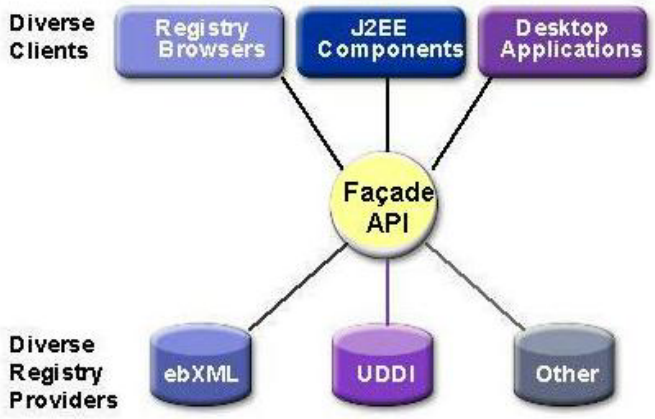


Figure 10 **Façade API between clients and registries.**

Comment [HJS(22]: Figure should be redone to include registry API layer.

7.2 CAPABILITY PROFILES

Because some diversity exists among registry provider capabilities, a multilayer API abstractions 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.

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

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 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 registry use cases can be considered level 2.

7.3 FAÇADE API DEFINITIONS

Comment [JAE23]: So are we using the JAXR API here or what? It seems you've introduced an abstract façade API that does not imply JAXR. If we are going to use JAXR, then we need to make it very clear that this is a Java-only based solution. – JSH: added note to clarify that the JAXR API is not required.

The following table presents a summarized list of the façade API's together with a brief description. The table also includes the source use cases and if available the CCSDS XML/Schema tool APIs highlighted. Annex 2 provides a mapping to the JAXR API, use cases, and CCSDS XML/Schema tool APIs. Note that the mapping from the Façade API to JAXR is for illustrative purposes only and does not imply that the use of the JAXR API is required.

Use Case	Client Façade API	Description	XML Schema API
Publish - This use case describes the actions necessary for a user to publish an artifact in the registry	**PublishObject**	**Publish a new information object in the registry.** 1. Validates the metadata (represesentation information) 2. Assign a unique identifier (with version) 3. Assign user provided local identifier (without version) 4. Update the catalog. 5. Store the data object in the repository. 6. Return the unique identifier. 7. Notification is sent to the subscribers	PublishSchema - Org
Version - This use case describes the actions necessary to version a new artifact	**VersionObject**	**Publish a new version of an information object** 1. Assign a unique identifier (including version) to the information object. 2. Local identifier of prior version is assigned to the information object.	
Store - This use case describes the actions necessary to store the artifact	**StoreObject**	**Store a new or updated information object (no validation, cataloging, or notification)** 1. Assign a unique identifier (with version) 2. Assign user provided local identifier (without version) 3. Store the data object in the repository. 4. Return the unique identifier.	
Catalog - This use case describes the actions necessary to catalog a new artifact	**CatalogObject**	**Update the catalog with the metadata**	
Validate - This use case describes the actions necessary to validate the metadata associated with an artifact	**ValidateObject**	**Validate an information objects representation data.**	
Update - This use case describes the actions necessary to update an artifact in the registry	**UpdateObject**	**Update information object.** 1. Replace the data object in the repository 2. Replace the metadata 3. Update the catalog 4. Notification is sent to the subscribers of the update	updateSchema - Org

Approve - This use case describes the actions necessary to approve an artifact in the registry	**ApproveObject**	**Find newly published information objects which are not already approved** 1. Update the metadata for an information object to indicate whether it is approved or rejected 2. Notification is sent to the publisher and subscribers	approveSchema - Org
Deprecate - This use case describes the actions necessary to deprecate an artifact in the registry	**DeprecateObject**	**Deprecate an information object** 1. System administrator updates the registry to indicate that a specific information object and version is deprecated. All associations are retained. 2. Subscribers are notified of the deprecated information object.	
Delete - This use case describes the actions necessary to delete an artifact in the registry	**DeleteObject**	**Delete an information object** 1. Delete the information object from the registry and the data object from the repository. All associations must have been removed prior to delete.	DeleteSchema - Org
Notify - This use case describes the actions necessary to subscribe to registry events	**Notify**	**Notify subscriber of registered event** 1. Notify the user when the event has occurred	
Discover - This use case describes the actions necessary to discover registered artifacts	**FindObject**	**Find an Information Object** 1. Get search criteria (generic) 2. Search the catalog 3. Return metadata (including pointers) describing registered information objects that meet the search criteria. Given a local identifier, return all versions.	
Discover	**FindAllAssociatedObject**	**Find Associated Information Objects** 1. Get search criteria (existing information object and association) 2. Search the catalog 3. Return metadata (including pointers) describing registered information objects that meet the search criteria. Given a local identifier, return all versions.	n/a

Discover	GetObject - Same as FindObject?	**Find an Information Object** 1. Get search criteria (generic) 2. Search the catalog 3. Return metadata (including pointers) describing registered information objects that meet the search criteria.Given a local identifier, return all versions.	getSchema - Org
Discover	GetObjectAssociatedObjects - Same as FindAllAssociatedObject	**Find an Associated Information Object** 1. Get search criteria (existing information object and association) 2. Search the catalog 3. Return metadata (including pointers) describing registered information objects that meet the search criteria. Given a local identifier, return all versions.	getSchemaAssociatedObjects - Org
Discover	GetAllObjectVersion - Subsumed by FindObject?	Subsumed	getAllSchemaVersions - Org
Retrieve - This use case describes the actions necessary to retrieve a registered artifacts	**GetRepositoryItem**	**Retieve Data Object** 1. Get one or more unique identifiers for information objects 2. Retrieve all the related data objects from the repository. Allow packaging options	getRepositoryItem - Org
Retrieve	**GetInformationObject** - Replaces GetObjectContentOfAllAssociateObjectsAsAZIPFile	**Retrieve Information Object** 1. Get one or more unique identifiers for information objects 2. Retrieve all Information Objects (representation information and data objects. Allow packaging options	getPackage - Org
Associate	**AddAssociation**	**Add Object Association** 1. Register an Association	
Associate	**Associate Objects** - Replaces AddObjectAssociatedObject	**Associate Objects** 1. Make an association between existing information objects A and B using an existing association.	addSchemaAssociatedObject - Org
Associate	**DeleteAssociation**	**Delete Object Association** 1. Delete an association. No associations to information objects are allowed.	
Associate	**DeleteAssociation BetweenObject**	**Delete Association Between Objects** 1. Delete an associiation between two or more information objects.	n/a
	n/a		findAllExtrinsicObjects - Org

Table 4 – Façade API Set

8 LIFECYCLE MANAGEMENT

8.1 OVERVIEW

This section[6] 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 [13].

Comment [JAE24]: Again, all code-like entries should be courier new font.

Formatted: Font: (Default) Courier New

8.2 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.

8.3 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

[6] Significant amounts of material have been extracted from the OASIS ebXML Registry Services and Protocols (ebXML RS) [13] and the ebXML Reference Information Model (ebXML RIM) [5] specifications for this section.

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

8.4 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.

8.5 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.

8.6 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.

8.7 REGISTRY MANAGED VERSION CONTROL

This section describes the version control features of the Registry.

Version Controlled Resources - All repository items in an 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.

ANNEX Sections

Annex 1 Reference Registry Use Cases

OPEN GIS PROJECT REGISTRY

The following table outlines general use cases that were produced for the OPEN GIS Project [4]. The details of the sub-use cases (e.g. description, pre-condition, and sequence of action) are provided in the referenced document.

Use Case Number	Use Case Name	Description (Sub-use cases)
GIS-1	Query metadata resource	1.1. Query metadata resource by identifier 1.2. Query metadata resource based on content 1.3. Query metadata resource by classification in taxonomy 1.4. Query metadata resource by responsible organization 1.5. Query metadata resource that is associated with other resource
GIS-2	Follow associations/links	2.1. Associate a registry or repository item with an externally located item 2.2. Associate a registry or repository item with an internally located item
GIS-3	Getting to classification	3.1. Find classification scheme/node by identifier 3.2. Find classification node by path expression 3.3. Find classification scheme by content 3.4. Classify a registry object by selected taxonomy
GIS-4	Federated registries	4.1. Administer a federation of registries 4.2. Perform distributed query with search policy
GIS-5	Publish	5.1. Publish dataset description to a registry (remotely reference content) 5.2. Publish service description to a registry (remotely reference content) 5.3. Publish service type to a registry (submit content) 5.4. Publish data type to a registry (submit content) 5.5. Publish taxonomy scheme to a registry 5.6. Publish style to a registry 5.7. Publish symbol set to a registry
GIS-6	Update	6.1. Update association between registry objects 6.2. Update registry object classification 6.3. Update classification scheme/node 6.4. Update registry object
GIS-7	Delete	7.1. Delete association between registry objects 7.2. Delete registry object classification 7.3. Delete classification scheme/node 7.4. Delete registry object
GIS-8	Deprecate	8.1. Deprecate classification scheme 8.2. Deprecate registry object
GIS-9	Security	9.1. Submit a publication request using the security mechanism

AMALFI MULTI-MISSIONS – XML SCHEMA REPOSITORY

The following use cases have been extracted from the AMALFI Multi Missions XML Schema Repository Technical Note (GAEL-P236-TCN-002).

A1.1.1.1 Actors

The actor specifies the role played by the users or any other system that interacts with the XML Schema Repository.

Any Actor models a type of role played by an entity that interacts with the XML Schema Repository (e.g. by exchanging data), but which is external to it i.e. in the sense that an instance of an actor is not a part of the instance of its corresponding repository. Actors may represent roles played by human users, external hardware, or other subjects. Note that an actor does not necessarily represent a specific physical entity but merely a particular role of some entity that is relevant to the specification of its associated use cases. Thus, a single physical instance may play the role of several different actors and, conversely, a given actor may be played by multiple different instances.

The following diagram and table introduce the actors that have been identified for the XML Schema Repository:

Actor	Description
User	Any entity that plays a role that interacts with the XML Schema Repository.
Human User	Any personnel interacting with the XML Schema Repository. Human User actors require command line or graphical interfaces with the XML Schema Repository e.g. shell commands or Web pages accessible in a Web client
Administrator	A specific Human User that has privileges for administrating the repository. The administrator may

	in particular install the repository; manage security level and user authentication/rights. An administrator may be seen as a "Super User".
Application	Any software interacting with the XML Schema Repository. In the current definition it is not foreseen that Application would be granted to perform administration tasks.
AMM	Any AMALFI Multi-Mission component: the targeted and main actor of the current project.

A1.1.1.2 XML Schema Submission Use Cases

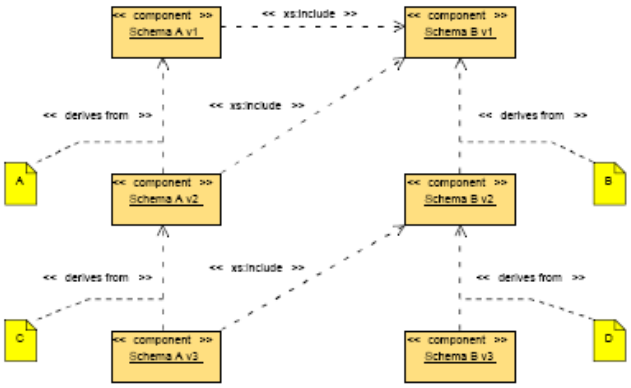
Comment [JAE25]: Just list the use case names organized by type like you did for the DISA case. Do not just show a Use Case Diagram (UCD) because that is an incomplete representation of the overall use case model. In fact, in all the examples you include in this annex, you should just list the use cases by name and call out the cited reference for the complete use case model (UCD, actors, use case specs).

Use Case Amalfi-1

Comment [JAE26]: Hard to read images (for all UCs in this section).

A1.1.1.3 Versioning Use Cases

Use Case Amalfi-2

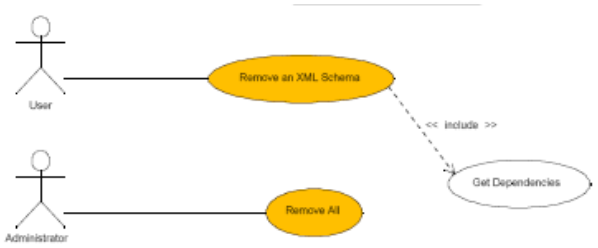


A1.1.1.4 XML Schema Retrieval Use Cases

Use Case Amalfi-3

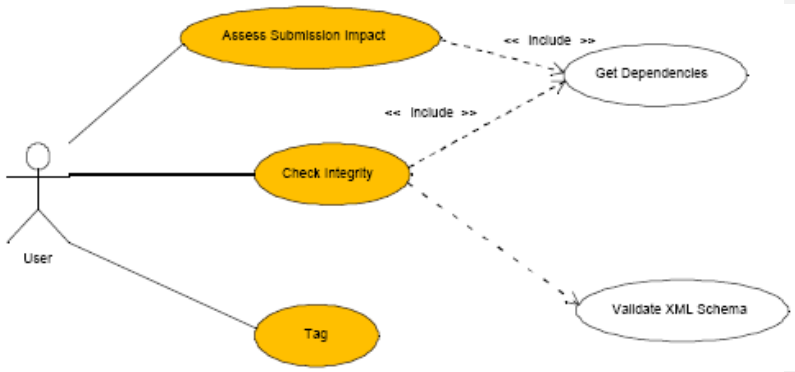
A1.1.1.5 XML Schema Removal Use Cases

Use Case Amalfi-4



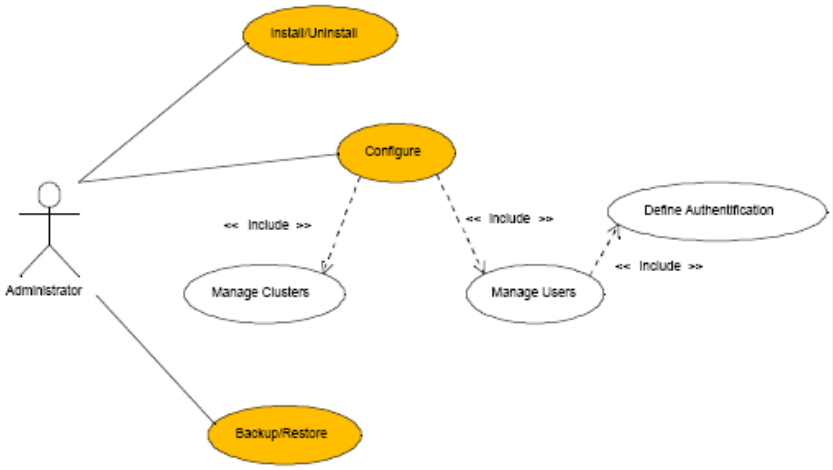
A1.1.1.6 Revision Control Use Cases

Use Case Amalfi-5



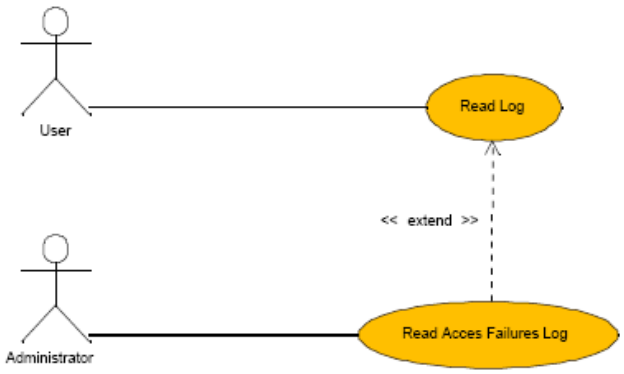
A1.1.1.7 Repository Administration Use Cases

Use Case Amalfi-6



A1.1.1.8 Logging/Monitoring Use Cases

Use Case Amalfi-7



JPL DEEP SPACE ~~NETWORK~~ INFORMATION SERVICES ARCHITECTURE (DISA) REGISTRY USE CASES

This section provides use case scenarios for the "DISA" registry.

A1.1.1.9 Background

The Deep Space ~~Network~~ Information Services Architecture (DISA) is a set of information services and information models to enable the Deep Space Network and Advance Multi-mission Operation System (AMMOS) to become a service-oriented architecture. As such, DISA has identified several services needed to support movement towards a SOA. A Registry Service is one such service that has identified needs for managing models, schemas, services, elements, and namespaces.

Comment [JAE27]: Need to state that this is an internal JPL initiative.

A1.1.1.10 Use Cases

A1.1.1.11 General Use Cases

Use Registered Service	Develop Schemas Collaboratively
Use Schema Versions	Perform XML Translation

Migrate Registry to Operations	

A1.1.1.12 Blanket Registry Use Cases

Backup Registry	Restore Registry
Provide Alternate Registry	

A1.1.1.13 Data Element Use Cases

Registry Element	Unregister Element
Update Element Metadata	Promote Element
List Elements	Query Elements
Get Element Metadata	

A1.1.1.14 XML Schema Registry Use Cases

Create Schema Directory	Register Schema
Unregister Schema	Update Schema
Update Schema Metadata	Build Elements from Schema
Promote Schema	List Schemas
Query Schemas	Get Schema Metadata
Get Versioned Schema	

A1.1.1.15 XML Stylesheet Use Cases

Create Stylesheet Directory	Register Stylesheet
Unregister Stylesheet	Update Stylesheet Metadata
Promote Stylesheet	List Stylesheets
Query Stylesheets	Get Stylesheet Metadata
Get Versioned Stylesheet	

A1.1.1.16 Namespace/Domain Registry Use Cases

Register Namespace	Unregister Namespace
Update Namespace Metadata	Promote Namespace
List Namespaces	Query Namespaces
Get Namespace Metadata	Locate Namespace Members

A1.1.1.17 Service Registry Use Cases

Register Service	Unregister Service
Update Service Metadata	Promote Service
Lookup Service	List Services
Get Service Metadata	Get Service Interface

CCSDS SERVICE LINK EXCHANGE (SLE) WG

A1.1.1.18 Background

The SLE Services provide a standard way of passing CCSDS telecommand and telemetry services across the ground segment. By implementing SLE services, TTC Services Providers will be able to provide a standard interface for supplying TTC services to Missions. This will reduce the cost of providing cross support services for spacecraft missions once the standard is in widespread use. In the near future, CCSDS tracking services and security will be added to the SLE capability, to facilitate the implementation of a fully operational SLE service.

A1.1.1.19 Use Cases

Schema Registry – registration and discovery of both SLE and cross-support XML schemas with notification
Data Elements – registration and access to common data elements used within CCSDS
Code Lists – common codes used within CCSDS
Services – agency published catalog of services as part of cross-support activities

CCSDS NAVIGATION WG

A1.1.1.20 Background

The Navigation Working Group provides a discipline-oriented forum for detailed discussions and development of technical flight dynamics standards.

A1.1.1.21 Use Cases

Schema Registry – registration and access to the navigation data messages for exchange of orbit representations, attitude representations, tracking data, general accelerations, etc within a federated environment
Data Elements – registration and access to common data elements used within CCSDS

CCSDS MISSION OPERATIONS SERVICES

A1.2 COMMON SM&C USE CASES

Comment [JAE28]: This is WAY TOO VERBOSE. You should only list the names of the use cases as you did for the DISA case. Assume this material exists in an external CCSDS document and it should be referred to rather than repeated.

Comment [JAE29]: Full of typos.

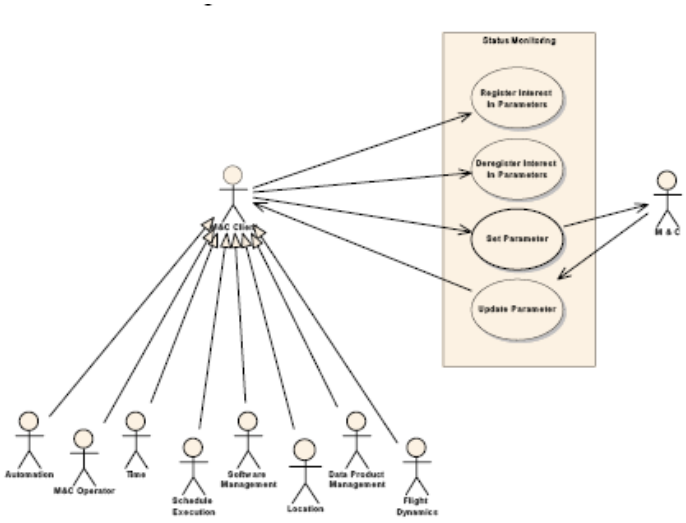
Use Case MOS-1	Register Interest
Brief description	Use case allows a user to register to receive updates It is espected that the registering of interest would involve the sending of some kind of filter and would also require some kind of privilege. The requet may also specify that only the current state should be supplied (single shot), or that the current state and subsequent changes in the state should be supplied (continuous)
Primary Actor	Client
Preconditions	The subsystem which provides the updates must be available. Client must have appropriate privileges to perform this.
Main Success Scenarios	1. The filter provided by th Client is validated 2. The Client is provided with the current state of all the items they have referenced with the subsystem to receive updates

Use Case MOS-2	Deregister Interest
Brief description	Use case allows a client to deregister interest in one or more items it previously registered for.
Primary Actor	Client

Preconditions	Was previously registered
Main Success Scenarios	The registration between the Client and the referenced items is removed

A1.3 CORE SM&C USE CASES

A1.3.1.1 Status Monitoring



Use Case MOS-3	**Register Interest In Parameters**
Brief description	Use case allows a user to register to receive updates reporting the state of one or more parameters. In this context the state of the parameter consists of all its dynamic attributes (e.g., value, status, raw data, quality flags) The request can specify that only the current parameter state should be supplied (single shot), or that the current

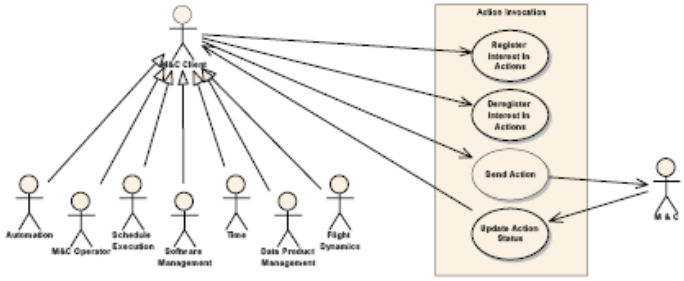
	parameter state and subsequent changes in the state should be supplied (continuous).
Primary Actor	M&C Client
Preconditions	The M&C Subsystem which provides the Parameter values must be available.
Main Success Scenarios	• All the parameter references provided by the M&C Status Client are validated. • The M&C Status Client is provided with the current state of all the parameters he has referenced in the request. • The M&C Status Client is registered with the Parameters.

Use Case MOS-4	**Deregister Interest In Parameters**
Brief description	Use case allows a user to deregister interest in one or more parameters.
Primary Actor	M&C Client
Preconditions	None
Main Success Scenarios	• All the parameter references provided by the M&C Status Client are validated. • The registration between the M&C Status Client and the referenced Parameters is removed.

Use Case MOS-4	**Set Parameters**
Brief description	Use case allows a Client to set a Parameter.

Primary Actor	M&C Client
Preconditions	The M&C Subsystem which maintains the Parameter values must be available.
Main Success Scenarios	• The Set request is forwarded to the M&C Subsystem

A1.3.1.2 Action Invocation



Use Case MOS-5	**Send Action**
Brief description	Use case allows a client to invoke an action (a symbolic control directive) by submitting an action request. The action request results in the creation of a new action instance, which is assigned a unique identifier. The action may be tagged for immediate execution, or tagged with an execution time.
Primary Actor	M&C Client
Preconditions	The target of the action must be available. The pre-transmission verification, if any, must be successful.
Main Success Scenarios	• The action is validated. • An action instance is created, and a unique

identifier allocated to it.

- The action is forwarded to the M&C Subsystem.
- The action is registered with the initiating client, so that the client will receive updates reporting changes in the action status.
- The action identifier is returned to the initiating client.

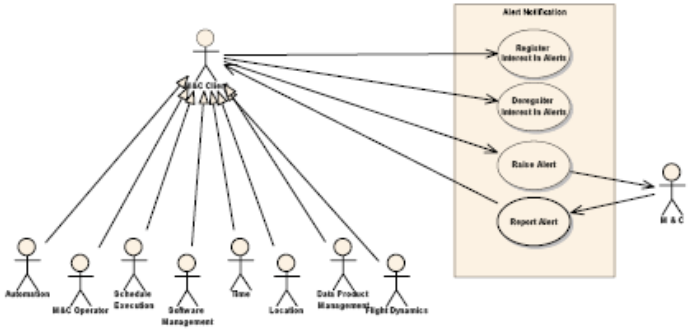
Use Case MOS- 6	**Update Action Status**
Brief Description	Use case allows a M&C System to report an update in the status of an Action.
Primary Actor	M&C (Subsystem)
Preconditions	The action must have been sent by a M&C Client.
Main Success Scenarios	• The action is validated. • The action status is reported all clients registered with the Action.

Use Case MOS-7	**Register Interest in Action**
Brief Description	Register to receive updates reporting status change of Actions. The Actions for which updates are required can be specified by any of the following methods :- Providing the instance identifiers – of the Actions for which updates are required. Providing the definition identifiers – of the Actions for which updates are required. Providing the Domain – updates are supplied for Actions executing in the Domain.
Primary Actor	M&C Client
Preconditions	None

Main Success Scenarios	• The Client is registered with the identified Actions. • For all Actions identified – report their current status to the client.

Use Case MOS-8	**Deregister Interest In Actions**
Brief Description	Use case allows a Space System to deregister for Action updates.
Primary Actor	M&C Client
Preconditions	None
Main Success Scenarios	• The registration between the Client and the Action is removed.

A1.3.1.3 Alert Notification



Use Case MOS-9	**Register Interest In Alerts**
Brief Description	Register to receive notification of Alerts. The Alerts for which notifications are required can

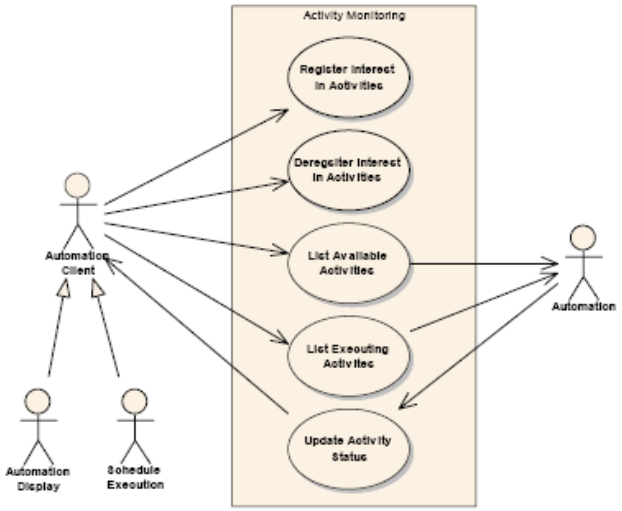
	be specified by any of the following methods : Providing the definition identifiers – of the Alerts for which updates are required. Providing the Domain – notifications are supplied for Alerts raised in the Domain.
Primary Actor	M&C Client
Preconditions	None
Main Success Scenarios	• The Client is registered with the Alert definition.

Use Case MOS-10	**Deregister Interest In Alerts**
Brief Description	Use case allows a Space System to deregister for Alerts.
Primary Actor	M&C Client
Preconditions	None
Main Success Scenarios	• The registration between the Client and the Alert definition is removed.

A1.4 OPERATIONS AUTOMATION USE CASES

A1.4.1.1 Activity Control Use Cases

A1.4.1.2 Activity Monitoring Use Cases



Use Case MOS-11	**Register Interest In Activities**
Brief Description	Register to receive updates reporting status change of Activities. The activities for which updates are required can be specified by any of the following methods : Providing the instance identifiers – of the activities for which updates are required. Providing the definition identifiers – of the activities for which updates are required. Providing the Domain – updates are supplied for activities executing in the Domain.
Primary Actor	Automation Client
Preconditions	None
Main Success Scenarios	• The Client is registered with the identified activities. • For all activities identified – report their current status to the client.

Use Case MOS-12	Deregister Interest In Activities
Brief Description	Deregister to receive updates reporting status change of Activities.
Primary Actor	Automation Client
Preconditions	Client has registered for the specified activities
Main Success Scenarios	• The registration between the Client and the activities is removed.

Use Case MOS-13	List Available Activities
Brief Description	List available activities.
Primary Actor	Automation Client
Preconditions	None
Main Success Scenarios	• Provide the Client with a list of all Activities that are

Use Case MOS-14	List Executing Activities
Brief Description	List executing activities.
Primary Actor	Automation Client
Preconditions	None
Main Success Scenarios	• Provide the Client with a list of all Activities that are currently executing.

Use Case MOS-15	Update Activity Status
Brief Description	Update the execution status of an activity
Primary Actor	Automation (Subsystem)
Preconditions	None
Main Success Scenarios	• Report the Activity Status to all Clients which have registered to receive updates for the Activity

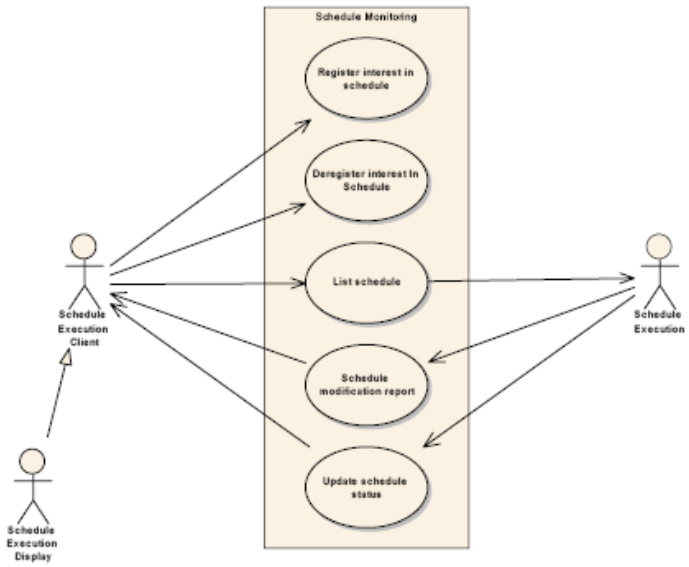
A1.5 OPERATIONS SCHEDULING USE CASES

A1.5.1.1 Schedule Level Control Use Cases

A1.5.1.2 Schedule Maintenance

A1.5.1.3 Schedule Activity Level Control Use Cases

A1.5.1.4 Schedule Monitoring Use Cases



Use Case MOS-16	Register Interest In Schedule
Brief Description	Register to receive updates reporting status change of schedule.
Primary Actor	Schedule Execution Client
Preconditions	None.
Main Success Scenarios	• The Client is registered with the schedule. • Report the current status to the client.

Use Case MOS-17	Deregister Interest In Schedule
Brief Description	Deregister to receive updates reporting status change of schedule.
Primary Actor	Schedule Execution Client

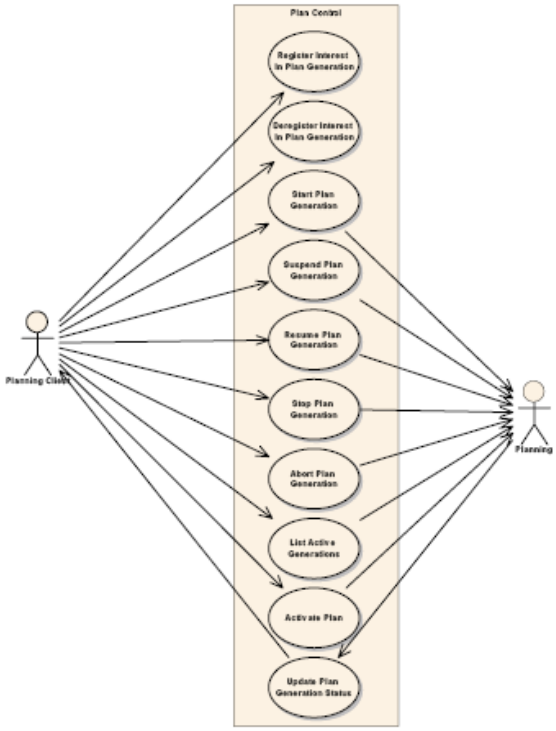
Preconditions	Client has registered.
Main Success Scenarios	• The registration between the Client and the schedule is removed.

Use Case MOS-18	**List Schedule**
Brief Description	List the schedule.
Primary Actor	Schedule Execution Client
Preconditions	None.
Main Success Scenarios	• Provide the Client with a list of all Activities that are currently contained in the schedule.

Use Case MOS-19	**Update Schedule Status**
Brief Description	Update the execution status of the schedule.
Primary Actor	Schedule Execution
Preconditions	None.
Main Success Scenarios	• Report the Schedule Status to all Clients which have registered to receive updates for the Schedule.

A1.6 OPERATIONS PLANNING USE CASES

A1.6.1.1 Planning Control Use Cases



Use Case MOS-20	Register Interest In Plan Generation
Brief Description	Allows the client to register for status updates for selected plan generation events.
Primary Actor	Planning Client
Preconditions	None.
Main Success Scenarios	• Client is registered for plan generation events. • Report the current status to the client.

Use Case MOS-21	Deregister Interest In Plan Generation

Brief Description	Allows a client to remove themselves from the list of clients to be notified of plan generation events.
Primary Actor	Planning Client
Preconditions	The client is previously registered.
Main Success Scenarios	• Client is no longer notified of plan generation events.

Use Case MOS-22	**List Active Plan Generations**
Brief Description	Returns the complete list of active plan generations.
Primary Actor	Planning Client
Preconditions	None.
Main Success Scenarios	• The list is returned.

Use Case MOS-23	**Update Plan Generation Status**
Brief Description	Notification to the client of a status change in the generation.
Primary Actor	Planning
Preconditions	Client has registered for updates.
Main Success Scenarios	• All registered client receive the update.

A1.6.1.2 Plan Level Maintenance Use Cases

A1.6.1.3 Plan Task Level Maintenance Use Cases

Use Case MOS-24	**Register Interest In Element Status**
Brief Description	Allows the client to register for status updates for selected plan elements. Plan elements include: Plans Tasks Activities Actions Constraints
Primary Actor	Planning Client
Preconditions	Elements exist.
Main Success Scenarios	• Client is registered for updates. • Report the current status to the client.

Use Case MOS-25	Deregister Interest In Element Status
Brief Description	Removes the client from receiving updates about the selected elements.
Primary Actor	Planning Client
Preconditions	Client is already registered to receive updates.
Main Success Scenarios	• Client is deregistered for updates.

Use Case MOS-26	Add Task
Brief Description	A task is added to a plan.
Primary Actor	Planning Client
Preconditions	None.
Main Success Scenarios	• Task is added.

Use Case MOS-27	Modify Task
Brief Description	An existing task is modified.
Primary Actor	Planning Client
Preconditions	Task exists.
Main Success Scenarios	• Task is modified.

Use Case MOS-28	Delete Task
Brief Description	Deletes a task from an existing plan.

Primary Actor	Planning Client
Preconditions	Task exists.
Main Success Scenarios	• Task is removed from the plan.

Use Case MOS-29	**Add Constraint**
Brief Description	Adds a constraint to a plan. A constraint can be on a element or between elements.
Primary Actor	Planning Client
Preconditions	Element being constrained exists.
Main Success Scenarios	• Constraint is inserted in plan.

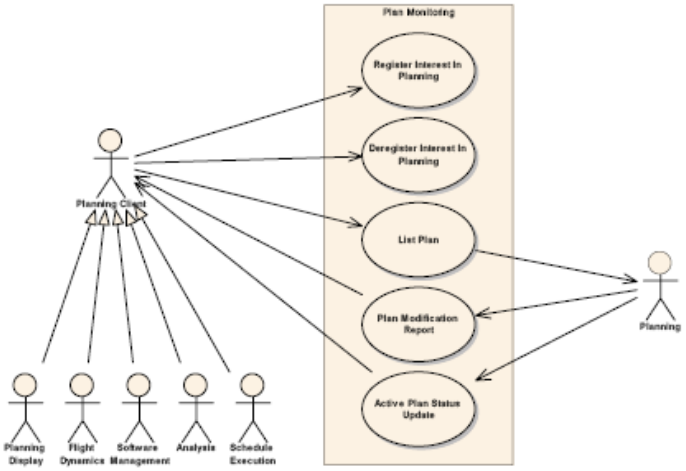
Use Case MOS-30	**Modify Constraint**
Brief Description	Modify an existing constraint.
Primary Actor	Planning Client
Preconditions	Constraint exists.
Main Success Scenarios	• Constraint is modified.

Use Case MOS-31	**Delete Constraint**

Brief Description	Delete an existing constraint from a plan.
Primary Actor	Planning Client
Preconditions	Constraint exists.
Main Success Scenarios	• Constraint is removed from the plan.

Use Case MOS-32	**Update Element Status**
Brief Description	Send notification of an update to an element to all registered clients.
Primary Actor	Planning
Preconditions	None.
Main Success Scenarios	• All registered clients are notified of update.

A1.6.1.4 Plan Monitoring Use Cases



Use Case MOS-33	**Register Interest In Planning**
Brief Description	Allows the client to register for status updates for selected planning events.
Primary Actor	Planning Client
Preconditions	None.
Main Success Scenarios	• Client is registered for planning events. • Report the current status to the client.

Use Case MOS-34	**Deregister Interest In Planning**
Brief Description	Allows a client to remove themselves from the list of clients to be notified of planning events.
Primary Actor	Planning Client
Preconditions	The client is previously registered.
Main Success Scenarios	• Client is no longer notified of planning

	events.

Use Case MOS-35	**List Plan**
Brief Description	Returns the complete plan, or a subsection of, the plan to the client.
Primary Actor	Planning Client
Preconditions	None.
Main Success Scenarios	• The requested plan is returned.

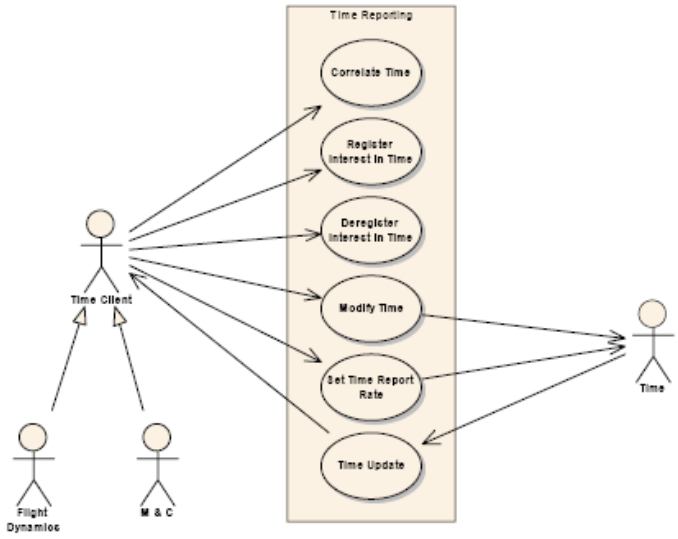
Use Case MOS-36	**Plan Modification Report**
Brief Description	Summary report of any modification made to the active plan.
Primary Actor	Planning
Preconditions	Client had registered for updates.
Main Success Scenarios	• All registered clients receive the update.

Use Case MOS-37	**Active Plan Status Update**
Brief Description	Notification to the client of a status change in the active plan.
Primary Actor	Planning

Preconditions	Client has registered for updates.
Main Success Scenarios	• All registered client receive the update.

A1.7 GUIDANCE, TRACKING AND SYNCHRONISATION USE CASES

A1.7.1.1 Time use cases



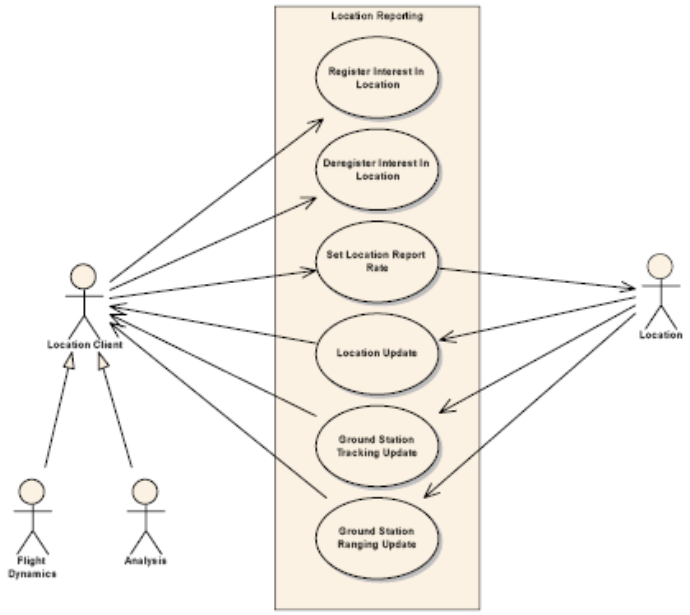
Use Case MOS-38	**Register Interest In Time**
Brief Description	Allows a client to register interest in time reports.
Primary Actor	Time Client
Preconditions	None.

Main Success Scenarios	• Client is registered for time reports.

Use Case MOS-39	**Deregister Interest In Time**
Brief Description	Allows a previously registered client to stop receiving time reports.
Primary Actor	Time Client
Preconditions	Client was previously registered.
Main Success Scenarios	• Client no long receives time reports.

Use Case MOS-40	**Time Update**
Brief Description	Time update is sent.
Primary Actor	Time
Preconditions	Client has registered for updates.
Main Success Scenarios	• All registered client receive a time update.

A1.7.1.2 Location Reporting use cases



Use Case MOS-41	**Register Interest In Location**
Brief Description	Allows a client to register interest in location reports.
Primary Actor	Location Client
Preconditions	None.
Main Success Scenarios	• Client is registered for location reports.

Use Case MOS-42	**Deregister Interest In Location**
Brief Description	Allows a previously registered client to stop receiving location reports.
Primary Actor	Location Client
Preconditions	Client was previously registered.

Main Success Scenarios	• Client no long receives location reports.

Use Case MOS-43	**Location Update**
Brief Description	Location update is sent.
Primary Actor	Location
Preconditions	Client has registered for updates.
Main Success Scenarios	• All registered client receive a location update.

Use Case MOS-44	**Ground Station Tracking Update**
Brief Description	Tracking update is sent.
Primary Actor	Location
Preconditions	Client has registered for updates.
Main Success Scenarios	• All registered client receive a tracking update.

Use Case MOS-45	**Ground Station Ranging Update**
Brief Description	Ranging update is sent.
Primary Actor	Location
Preconditions	Client has registered for updates.
Main Success Scenarios	• All registered client receive a ranging

	update.

A1.7.1.3 Location Control use cases

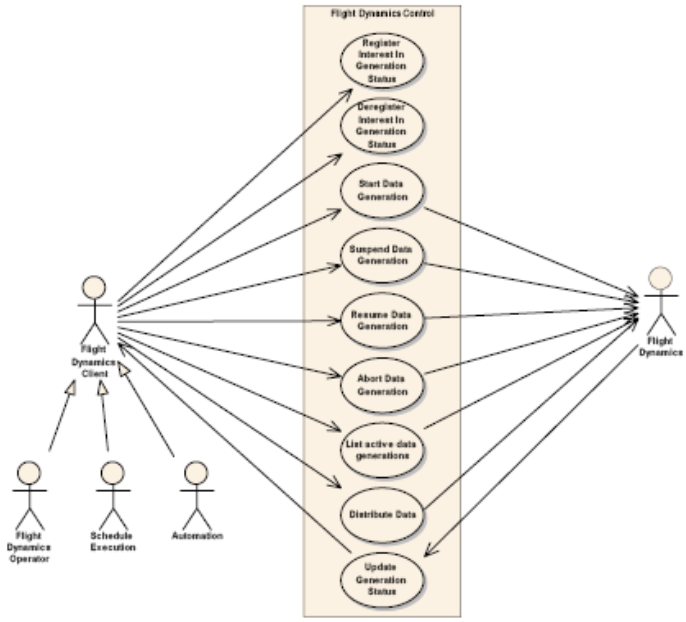
Use Case MOS-46	**Register Interest In Location Control**
Brief Description	Allows a client to register interest in location control updates.
Primary Actor	Location Client
Preconditions	None.
Main Success Scenarios	• Client is registered for location control updates.

Use Case MOS-47	Deregister Interest In Location Control
Brief Description	Allows a previously registered client to stop receiving location control updates.
Primary Actor	Location Client
Preconditions	Client was previously registered.
Main Success Scenarios	• Client no long receives location control updates.

Use Case MOS-48	List Active Ranging and Tracking
Brief Description	Provide the client with a list of active operations and their status.
Primary Actor	Location Client
Preconditions	None.
Main Success Scenarios	• Provide the Client with a list of all operations that are currently active.

Use Case MOS-49	Update Ranging and Tracking Status
Brief Description	Ranging or Tracking update is sent.
Primary Actor	Location Client
Preconditions	Client has registered for updates.
Main Success Scenarios	• All registered clients receive the update.

A1.7.1.4 Flight Dynamics Control use cases



Use Case MOS-50	Register Interest In Generation Status
Brief Description	Allows a client to register interest in flight dynamics generation status.
Primary Actor	Flight Dynamics Client
Preconditions	None.
Main Success Scenarios	• Client is registered for generation status notification. • Report the current status to the client.

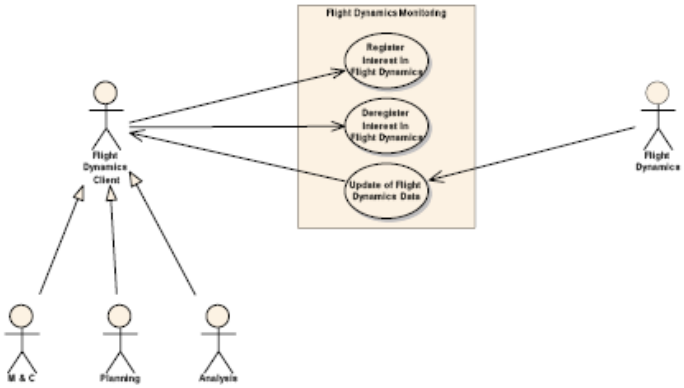
Use Case MOS-51	Deregister Interest In Generation Status
Brief Description	Allows a previously registered client to stop receiving flight dynamics generation status.

Primary Actor	Flight Dynamics Client
Preconditions	Client was previously registered.
Main Success Scenarios	• Client no long receives generation status notification.

Use Case MOS-52	**List Active Data Generations**
Brief Description	Provide the client with a list of active data generations and their status.
Primary Actor	Flight Dynamics Client
Preconditions	None.
Main Success Scenarios	• Provide the Client with a list of all generations that are currently executing.

Use Case MOS-53	**Update Generation Status**
Brief Description	Informs clients registered for updates of a change in the state of a generation task.
Primary Actor	Flight Dynamics
Preconditions	Client is registered.
Main Success Scenarios	• Report the generation to all Clients which have registered to receive updates.

A1.7.1.5 Flight Dynamics Monitoring use cases



Use Case MOS-54	**Register Interest In Flight Dynamics**
Brief Description	Allows a client to register interest in flight dynamics data. The registration allows the client to select the type of data it wants to receive. Data items include: • Orbit vectors • Ground station visibilities • Predicted events • Antenna steering data • Attitude data • Physical state data • Manoeuvre control data • Fuel budget assessment • End of Life prediction
Primary Actor	Flight Dynamics Client
Preconditions	None.
Main Success Scenarios	• Client is registered for data notification. • Report the current status to the client.

Use Case MOS-55	Deregister Interest In Flight Dynamics
Brief Description	Allows a previously registered client to stop receiving flight dynamics data.
Primary Actor	Flight Dynamics Client
Preconditions	Client was previously registered.
Main Success Scenarios	• Client no long receives data.

Use Case MOS-56	Update Of Flight Dynamics Data
Brief Description	A new version of a data files has been distributed.
Primary Actor	Flight Dynamics
Preconditions	Client is registered.
Main Success Scenarios	• Send the data to all Clients which have registered to

A1.8 REMOTE SOFTWARE MANAGEMENT USE CASES

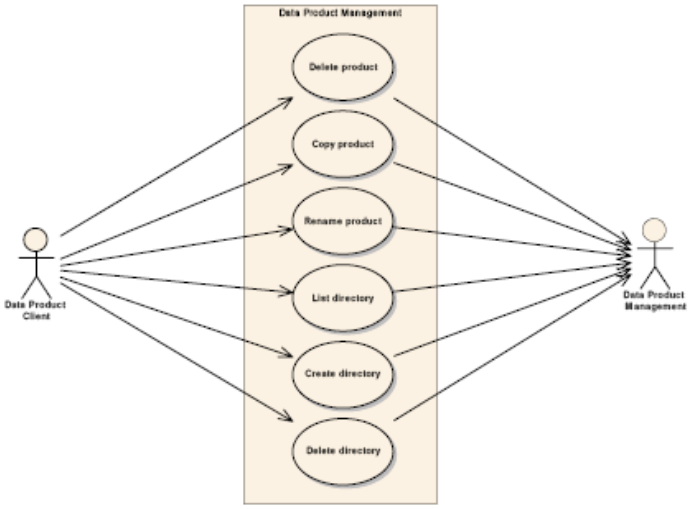
Use Case MOS-57	Register Interest In Software Management
Brief Description	Allows a client to register interest in Software Management data.
Primary Actor	Software Management Client
Preconditions	None.
Main Success Scenarios	• Client is registered for data notification.

Use Case MOS-58	Deregister Interest In Software Management
Brief Description	Allows a previously registered client to stop receiving Software Management data.
Primary Actor	Software Management Client
Preconditions	Client was previously registered.
Main Success Scenarios	• Client no long receives data.

Use Case MOS-59	**Software Check Report**
Brief Description	Reports the result of a software check.
Primary Actor	Software Management
Preconditions	Client is registered.
Main Success Scenarios	• Send the data to all Clients which have registered to receive it.

A1.9 PAYLOAD DATA PRODUCT MANAGEMENT USE CASES

A1.9.1.1 Data Product Management use cases



Use Case MOS-60	Delete Product
Brief Description	Deletes a product from the remote data store.
Primary Actor	Data Product Client
Preconditions	Product exists in the store.
Main Success Scenarios	• Product is deleted.

Use Case MOS-61	Copy Product
Brief Description	Copies a product in the remote data store.
Primary Actor	Data Product Client
Preconditions	Product exists in the store.
Main Success Scenarios	• Product is copied.

Use Case MOS-62	Rename Product
Brief Description	Renames a product in the remote data store.
Primary Actor	Data Product Client
Preconditions	Product exists in the store.
Main Success Scenarios	• Product is renamed.

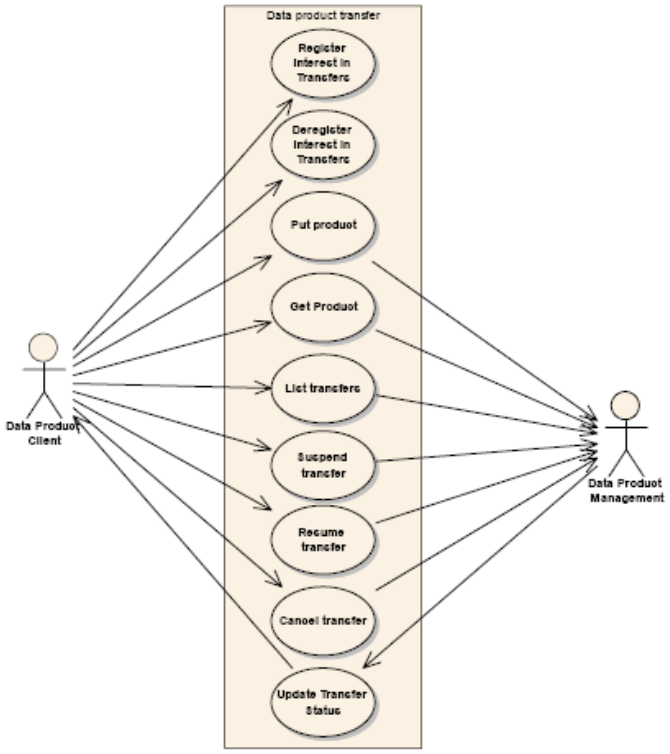
Use Case MOS-63	List Directory
Brief Description	Returns a list of files in the specified directory.

Primary Actor	Data Product Client
Preconditions	Directory exists in the store.
Main Success Scenarios	• Directory list is returned.

Use Case MOS-64	**Create Directory**
Brief Description	Creates a directory from the remote data store.
Primary Actor	Data Product Client
Preconditions	Parent directory exists in the store. No existing directory or file with the same name.
Main Success Scenarios	• Directory is created.

Use Case MOS-65	**Delete Directory**
Brief Description	Deletes a directory from the remote data store.
Primary Actor	Data Product Client
Preconditions	Directory exists in the store and is empty.
Main Success Scenarios	• Directory is deleted.

A1.9.1.2 Data Product Transfer use cases



Use Case MOS-66	Register Interest In Transfers
Brief Description	Allows a client to register interest in data transfer status updates.
Primary Actor	Data Product Client
Preconditions	None.
Main Success Scenarios	• Client is registered for data notification. • Report the current status to the client.

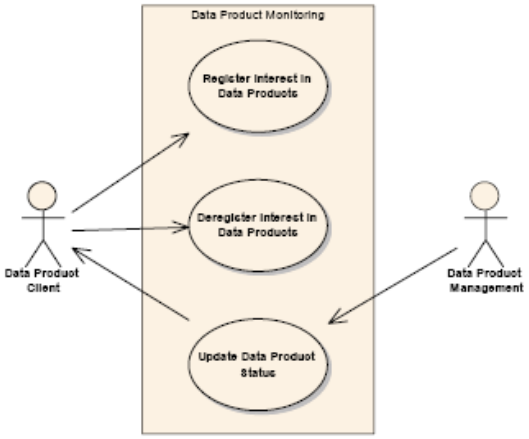
Use Case MOS-67	Deregister Interest In Transfers

Brief Description	Allows a previously registered client to stop receiving status updates.
Primary Actor	Data Product Client
Preconditions	Client was previously registered.
Main Success Scenarios	• Client no longer receives status updates.

Use Case MOS-68	**List Transfers**
Brief Description	Provide the client with a list of active data transfers and the status of them.
Primary Actor	Data Product Client
Preconditions	None.
Main Success Scenarios	• Provide the Client with a list of all transfers that are currently executing.

Use Case MOS-69	**Update Transfer Status**
Brief Description	Informs clients registered for updates of a change in the state of a transfer.
Primary Actor	Data Product Management
Preconditions	Client is registered.
Main Success Scenarios	• Report the transfer state change to all Clients which have registered to receive data product updates.

A1.9.1.3 Data Product Monitoring use cases

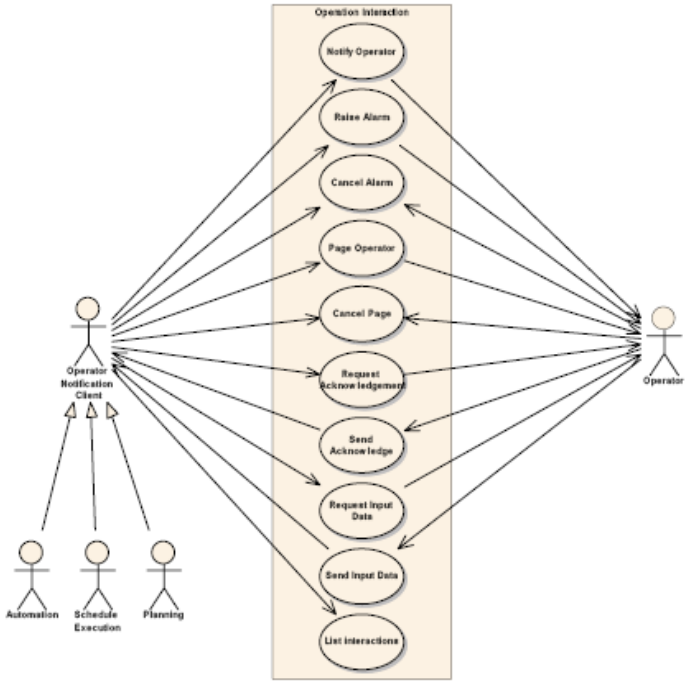


Use Case MOS-70	Register Interest In Data Products
Brief Description	Allows a client to register interest in data product status updates.
Primary Actor	Data Product Client
Preconditions	None.
Main Success Scenarios	• Client is registered for data product notification.

Use Case MOS-71	Deregister Interest In Data Products
Brief Description	Allows a previously registered client to stop receiving product updates.
Primary Actor	Data Product Client
Preconditions	Client was previously registered.
Main Success Scenarios	• Client no long receives product status updates.

Use Case MOS-72	Update Data Product Status
Brief Description	Informs clients registered for updates of a change in the state of a data product.
Primary Actor	Data Product Management
Preconditions	Client is registered.
Main Success Scenarios	• Report the data product state change to all Clients which have registered to receive updates.

A1.10 OPERATOR INTERACTION USE CASES



Use Case MOS-73	Notify Operator
Brief Description	Used to send a message to an operator that does not require acknowledgement.
Primary Actor	Operator Notification Client
Preconditions	None.
Main Success Scenarios	• Operator is notified.

Use Case MOS-74	Notify Operator
Brief Description	Used to send a message to an operator that does not require acknowledgement.
Primary Actor	Operator Notification Client

Preconditions	None.
Main Success Scenarios	• Operator is notified.

Use Case MOS-75	**Request Input Data**
Brief Description	Request input data from an operator. This is separate from normal operator interaction with client application, it is expected that normally autonomous systems shall use it when operator interaction is required. Several input methods are likely to be supported: – String – Number – Select option
Primary Actor	Operator Notification Client
Preconditions	None.
Main Success Scenarios	• Input is requested from operator

Use Case MOS-76	**Send Input Data**
Brief Description	Sends the input data that was requested previously.
Primary Actor	Operator
Preconditions	Input data was requested.
Main Success Scenarios	• Input data is sent.

Use Case MOS-77	**List Interactions**

Brief Description	Provide the client with a list of active interactions and the status of them.
Primary Actor	Operator Notification Client
Preconditions	None.
Main Success Scenarios	• Provide the Client with a list of all interactions that are currently outstanding.

Annex 2 JAXR APIs Mappings

The following table provides the mapping for the JAXR API set to the use cases of Chapter 4 and to the CCSDS XML/Schema tool APIs.

API	Description	Use Cases	XML Schema APIs
addRegistryObjects	Adds RegistryObjects.	publishObject	ingestSchema
getRegistryObjects	Gets the collection of member RegistryObjects of this RegistryPackage.		getPackage, getRepositoryItemForLatestVersion, getSchema
removeRegistryObjects	Removes RegistryObject.	deleteObject	deleteObject (deleteSchema)
addAssociation	Adds specified Association for this object.	addObjectAssociation	
addClassifications	Adds specified Classification to this object.		
addExternalIdentifier	Adds specified ExternalIdentifier as an external identifier to this object.		
addExternalLink	Adds specified ExternalLink to this object.		
getAssociatedObjects	Returns the collection of RegistryObject instances associated with this object.	getObjectAssociation	getSchemaAssociatedObjects
getAssociations	Gets all Associations where this object is the source.	getObjectAssociation	
getAuditTrail	Returns the complete audit trail of all requests that effected a state change in this object		
getClassifications	Get the Classification instances that classify this object.		
getDescription	Get the textual description for this object.		
getExternalIdentifiers	Returns the ExternalIdentifiers associated with this object.		
getExternalLinks	Returns the ExternalLinks associated with this object.		
getKey	Gets the key representing the universally unique ID (UUID) for this object.		
getLifeCycleManager	Returns the LifeCycleManager that created this object.		
getName	Gets the user-friendly name of this object.		
getObjectType	Gets the object type that best describes the RegistryObject.		
getRegistryPackages	Returns the Package associated with this object.	getObject	
getSubmittingOrganization	Gets the Organization that submitted this RegistryObject.		
removeAssociation	Removes specified Association from this object.		
removeClassification	Removes specified Classification from this object.		deleteAssociationsForObject
removeExternalIdentifier	Removes specified ExternalIdentifier as an external identifier from this object.		

removeExternalLink	Removes specified ExternalLink from this object.		
setAssociations	Replaces all previous Associations from this object with specified Associations.		
setClassifications	Replaces all previous Classifications with specified Classifications.		addSchemaAssociatedObject
setDescription	Sets the context independent textual description for this object.		
setExternalIdentifiers	Replaces all previous external identifiers with specified Collection of ExternalIdentifiers as an external identifier.		
setExternalLinks	Replaces all previous ExternalLinks with specified ExternalLinks.		
setKey	Sets the key representing the universally unique ID (UUID) for this object.		createObjKey
setName	Sets user-friendly name of object in repository.		
toXML	Returns a registry provider specific XML representation of this Object.		
		findObject	findLatestVersionKey
		setCredentials	findAllExtrinsicObjects
		updateObject	getAllSchemaVersions
		connect	getObjectContentIfExists
			getRepositoryItem
			getSchemaImportsIncludes
			getSchemaRepositoryObects
			publishTargetNamespace
			updateSchema
			validate

Annex 3 Electronic Business using XML Registry (ebXML Registry) [7]

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

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

• 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 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..

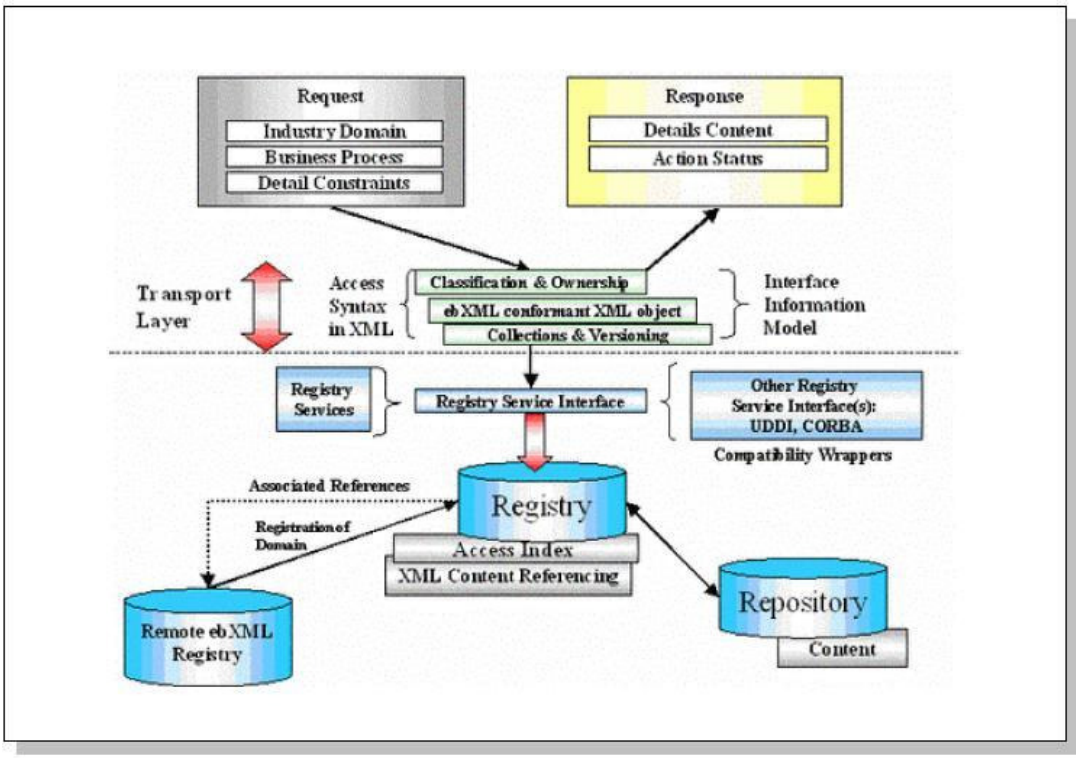


Figure 1 - ebXML Technical Archictecture

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).