Consultative Committee for Space Data Systems

RECOMMENDATION FOR SPACE DATA SYSTEM STANDARDS

Registry and Repository Reference Model

CCSDS [number]

DRAFT WHITE BOOK

April 7, 2009

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

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 and 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 W3C and Web Services standards to provide advanced data access techniques and adherence to the OAIS Reference Model (reference [7]) information model.

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, including an account of informational items that are used in large-scale information systems. These items range from data values such as names and codes, to vocabularies, services and software components. Within the business community, a registry has been defined as "an information system that securely manages any content type and the standardized metadata that describes it." [6]

This paper presents a set of use cases for registries. These use cases describe sequences of actions for specific purposes between a registry and its users. These uses case in turn will be used to derive requirements and help define classification attributes.

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 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 to another, and event notifications.

- Section 6 presents a model for extrinsic objects. Since the registry 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 operations over a diverse set of registries and defines a unified information model for describing registry contents. Regardless of the registry provider, applications use common APIs and a common information model.

- Section 8 defines the protocols supported by 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.

Annexes 1-2

- 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.

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.

1.5 REFERENCES

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

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

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

[5] OASIS/ebXML Registry Information Model v2.0, Approved OASIS Standard, OASIS/ebXML Registry Technical Committee, April 2002

[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

2 OVERVIEW OF A SERVICE 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 provides the following essential functions:

- 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, these features extend over the federated registries. The use of existing standards, such as 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. Finally access control and data management modules, tools, and a governance model bridge the functionality gap to support the enterprise applications.

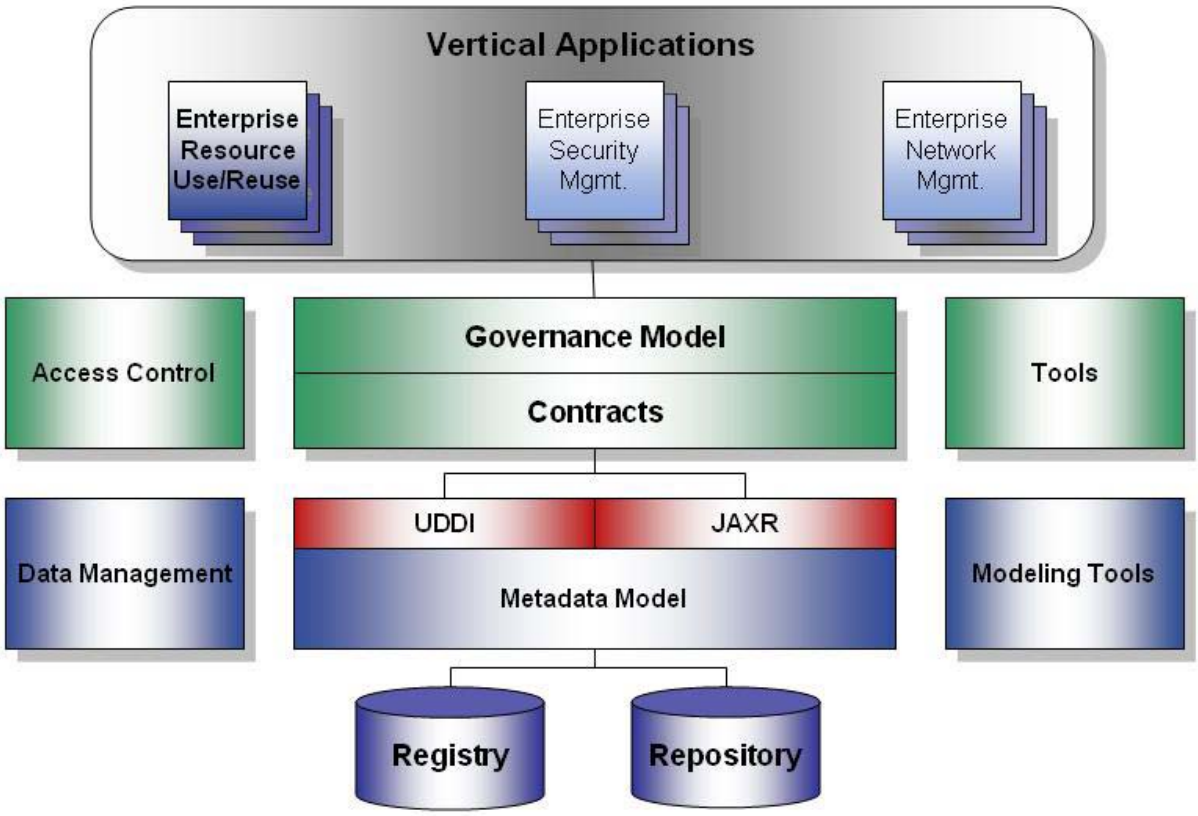


Figure 1 Environment View of a Registry and Repository

2.2 FUNCTIONAL VIEWS OF A REGISTRY AND REPOSITORY

A registry allows organizations to publish and discover both Web services and generic artifacts. The two dominate registry standards, UDDI (Universal Description, Discovery, and Integration) and ebXML (Electronic Business using eXtensible Markup Language) illustrate that registry support for Web services differs from the support required for generic artifacts. For example where ebXML supports services as generic artifacts, it in addition provides service bindings for the registered services.

Two functional views result, a registry as a Service Address Book or as an Information Repository. These two functional views 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 service.

This document focuses on a generic reference model for a registry, where services and generic artifacts are managed in the same way, as much as possible.

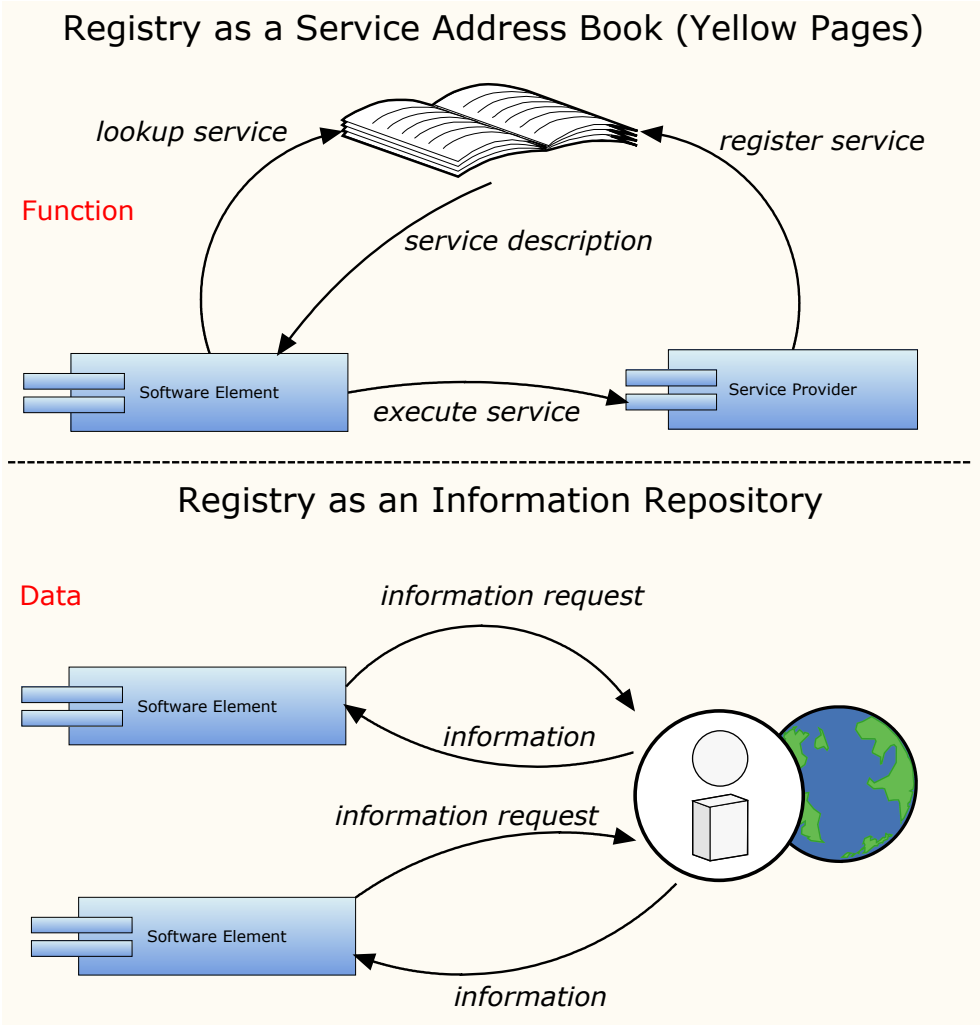


Figure 2: Two Conceptual Views of a Registry/Repository

2.3 GENERAL FEATURES OF A REGISTRY AND REPOSITORY

A registry and repository need to support the registration and discovery of data artifacts and services by providing interfaces for their submission, approval, and publishing, 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 Application Programming Interface (API) and Information Model sections of this paper describe the individual APIs and model classes that support the functionalities associated with Content Management, Events, Secure Architecture, and Web Services Registry. The Federation section of this document describes the Federated Architecture.

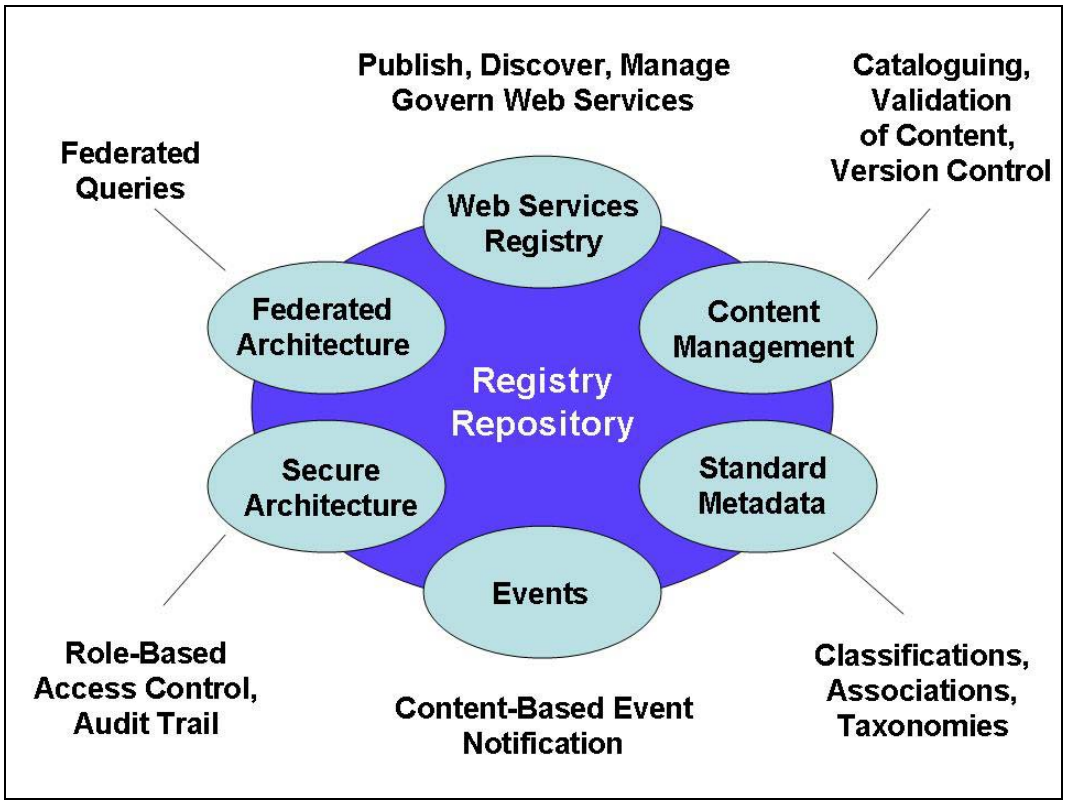


Figure 3: Features of a Federated Registry/Repository

3 USE CASES

The purpose of this section is to capture use case scenarios for registries. These use cases have been derived from several sources including the Information Architecture paper [1] , and formally defined Use Cases for the DSMS Information Services Architecture Registry [7], Open GIS Project Registry, AMALFI Multi-Missions – Xml Schema Repository, JPL Deep Space Network Information Service Architecture Registry, CCSDS Service Link Exchange (SLE) WG, CCSDS Navigation WG, Common and Core SM&C, Operations Automation and Scheduling, Remote Software Management, Payload Data Product Management, and Operator Interaction.

3.1 ACTORS

The following actors 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

artifact consumer <<system or person>> - An artifact consumer is a system or person that receives an artifact from a registry

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.

system administrator <<person>> - A system administrator is a person employed to maintain, and operate a registry/repository configuration.

registry service <<system>> - A registry service is a system interface through which another actor is able to perform registry 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 in the most general cases. Since general use cases have been developed by many other tasks, the intent here is to convey a general concept of actors and functions that are supported by registries. 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 services [8], the interface available for performing registry functions.

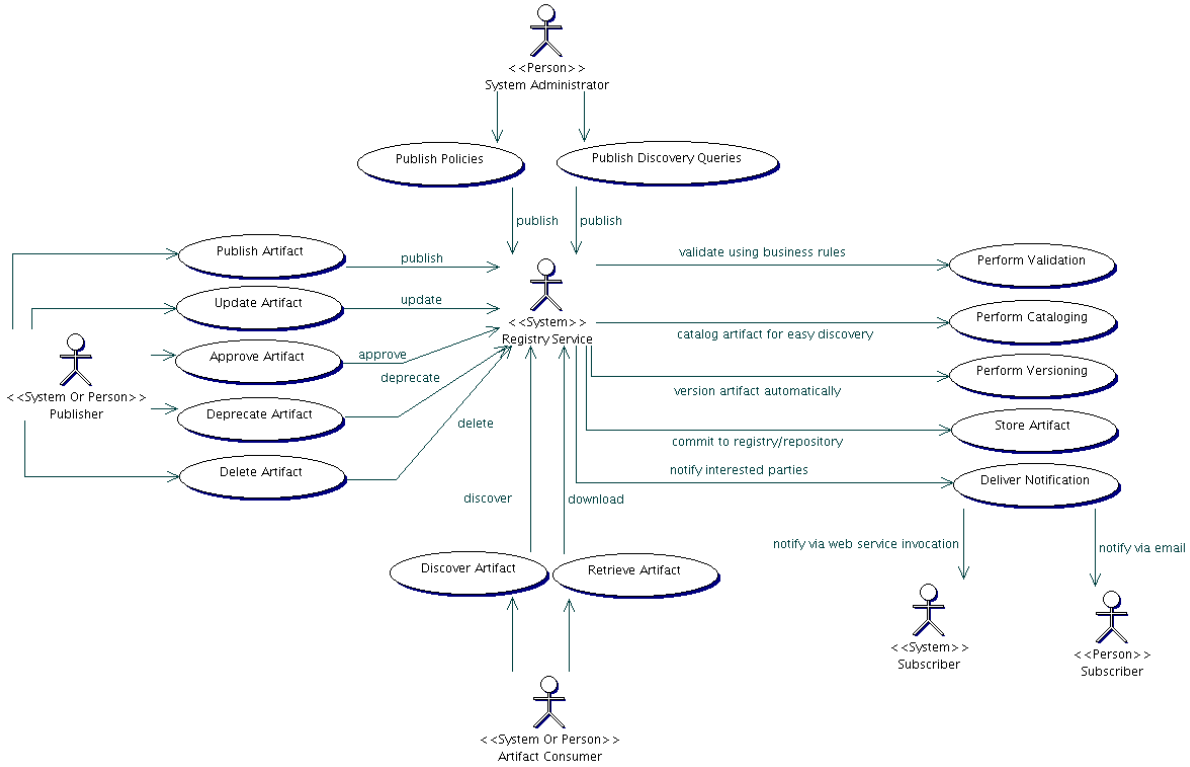


Figure 4 General Registry Repository Use Cases

In Annex 2, the general use cases in Figure 4 are mapped to the use cases that were produced for the OPEN GIS Project [4].

3.2.1 PUBLISH

Description

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

Actors: Registry Publisher, Registry Service

Actions:

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

3.2.2 UPDATE

Description

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

Actors: Registry Publisher, Registry Service

Actions:

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

3.2.3 APPROVE

Description

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

Actors: System Administrator, Registry Service

Actions:

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

3.2.4 DEPRECATE

Description

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

Actors: System Administrator, Publisher, Registry Service

Actions:

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.

3.2.5 DELETE

Description

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

Actors: Registry Publisher, Registry Service

Actions:

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

3.2.6 VALIDATE

Description

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

Actors: Registry Service

Actions:

1. User publishes a new artifact which includes descriptive metadata about the artifact
2. Registry service validates the metadata

3.2.7 CATALOG

Description

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

Actors: Registry Service

Actions:

1. User publishes a new artifact which includes descriptive metadata about the artifact
2. Registry 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: Registry Service

Actions:

1. User publishes a new artifact
2. Registry 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: Registry Service

Actions:

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

3.2.10 NOTIFY

Description

This use case describes the actions necessary to subscribe to registry events

Actors: Registry Consumer, Registry Service

Actions:

1. User creates a subscription for the registered event
2. Registry 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, Registry Service

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 a registered artifacts

Actors: Registry Consumer, Registry Service

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. 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 actions

described in 4.2. At present, these are specifically service and XML schema registry use cases.

3.4.1 SERVICE REGISTRY 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 allowing for registration and discovery of online services.

3.4.1.1 Actors

Service registry – a specialized registry for the management and discovery of online services

Service provider - A service provider is a system or person that provides a service to be submitted into the registry

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

3.4.1.2 Service Registration Use Cases

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

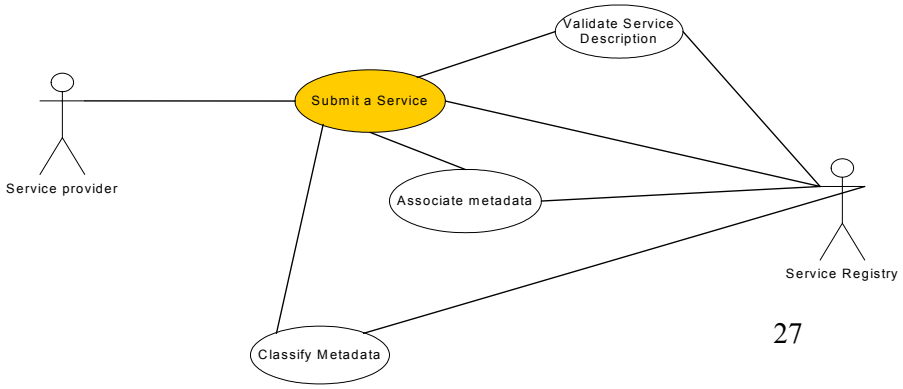
Actors: Service registry, Service Provider

Scenarios:

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

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

3. The registry service allows classification of the registered service



based on namespace

4. The registry 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

Actors: Service Registry, Service Consumer

Scenarios

1. A service consumer requests information about registered services from the service registry

2. The registry 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

Actors: Service Provider, Service Consumer

Scenarios

1. A service provider requests the service registry to remove the service

2. The registry service removes the entry from the catalog

3.4.2 XML SCHEMA REGISTRY

This section provides use cases for an XML Schema Registry. An XML Schema Registry allows for the registration, discovery and management

of XML schemas. The XML schema registry inherits several of functions of the general registry described earlier.

3.4.2.1 Actors

XML schema registry service – a specialized registry for the management and discovery of XML schemas

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

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

3.4.2.2 XML Schema Registration Use Cases

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

Actors/Users: XML Schema Registry, XML Schema Provider

Scenarios

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

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

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

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

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

6. The registry 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 service.

Actors/Users: XML Schema Registry, XML Schema Consumer

Scenarios

1. A XML Schema Consumer searches the XML schema registry based on a set of criteria.

2. The registry 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 Registry, XML Schema Consumer

Scenarios

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

2. The registry 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

Actors/Users: XML Schema Registry, XML Schema Consumer

Scenarios

1. A XML schema registry consumer provides an XML document to the XML Registry Service requesting that it be validated against a registered XML schema based on the unique identifier

2. The registry 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 provider updates the metadata for a XML schema in the registry

Actors: XML Schema Provider, XML Schema Registry

Scenarios

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

2. The registry 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 provider registers a new version of a schema

Actors: XML Schema Provider, XML Schema Registry

Scenarios

1. A XML Schema Provider registers a new version of a XML schema file through a registry service using standardized metadata.

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

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

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

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

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

3.4.2.8 XML Schema Removal Use Cases

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

Actors: XML Schema Provider, XML Schema Registry

Scenarios

1. A XML schema provider requests the XML Schema Registry to remove the schema based on a unique identifier

2. The registry service removes the schema from the repository

3. The registry service removes the description from the catalog

4 INFORMATION MODEL

The CCSDS Registry Information Model specification document [12] defines the general registry information model. This information model, based on the ebXML Reference Information Model (RIM) [5], is assumed if the JAXR (Java API for XML Registries) standard API is adopted for CCSDS registries. Assuming that the JAXR API is adopted, then the ebXML RIM provides a benchmark information model that can be used to develop a general information model for CCSDS Registries that can be extended for specific implementation, for example an XML Schema registry. The ebXML RIM subsumes the UDDI information model.

4.1 OVERVIEW OF A REGISTRY INFORMATION MODEL

A registry allows organizations to publish and discover Web services and artifacts. Currently, two registry standards dominate: UDDI (Universal Description, Discovery, and Integration) and ebXML (Electronic Business using eXtensible Markup Language). With either of these, science organizations can publish Web services and artifacts and their internal or external participating organizations can discover them.

The registry information model defines the classes and associations that support the registry features illustrated in Figure 3 such as Content Management, Events, Secure Architecture, Web Services Registry and the Federated Architecture. The objects acted on by the registry API are defined in the information model and support the required functionality such as Publish, Discover, and Manage registry objects.

4.2 A GENERAL REGISTRY INFORMATION MODEL

A unified information model for describing registry contents is defined by the JAXR (Java API for XML Registries) standard API. The information defines the classes necessary for publication and discovery of Web services and artifacts in the underlying registries.

The ebXML Registry Information Model (RIM) [5] defines the JAXR unified information model. The Registry Information Model provides a blueprint or high-level schema for an ebXML Registry. It provides implementers with information on the type of metadata that is stored in the Registry as well as the relationships among metadata Classes. The Registry information model defines what types of objects are stored in the Registry and how stored objects are organized in the Registry. The current specification leverage as much as possible the work done in the OASIS [OAS] and the ISO 11179 [ISO] Registry models.

4.2.1 RESPONSE TO USE CASES

The following table provides the information model response to the registry use cases. For each use case, the associated actions are used to derive functional requirements for registry functions. From the functional requirements the information model classes and the instantiated objects needed by the registry 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 Functional Requirements** -- Provide services and tools that implement system functions.	**Derived Requirements affecting Data Architecture** -- define standard models that support System Services.	**Suggested Class**
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, Regist ClassificationSc Identifiable, ContentInformat Person, Organiz Service, Notifica Subscription, Us VersionInfo, Rep**
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		

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**	**QueryExpressio** **StatusType, pub**
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 adminitr**
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**	**RegistryPackag** **ExtrinsicObject**

Table 1 – Derived Classes

4.2.2 VIEWS OF THE REGISTRY MODEL

The CCSDS Registry Information Model Specification document [12] provides a formal data engineering definition of the registry information model captured from the ebXML Registry Information Model (RIM) specification. [5] In the following two sections conceptual and logical views of the formation model are provided.

4.2.2.1 Conceptual

A conceptual model defines the community model from a manager's point of view. It 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 below.

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

Table 1 - CCSDS Registry Components and Functions

The following Conceptual Map illustrates these key concepts and their relationships as well as additional concepts needed to support the essential functions.

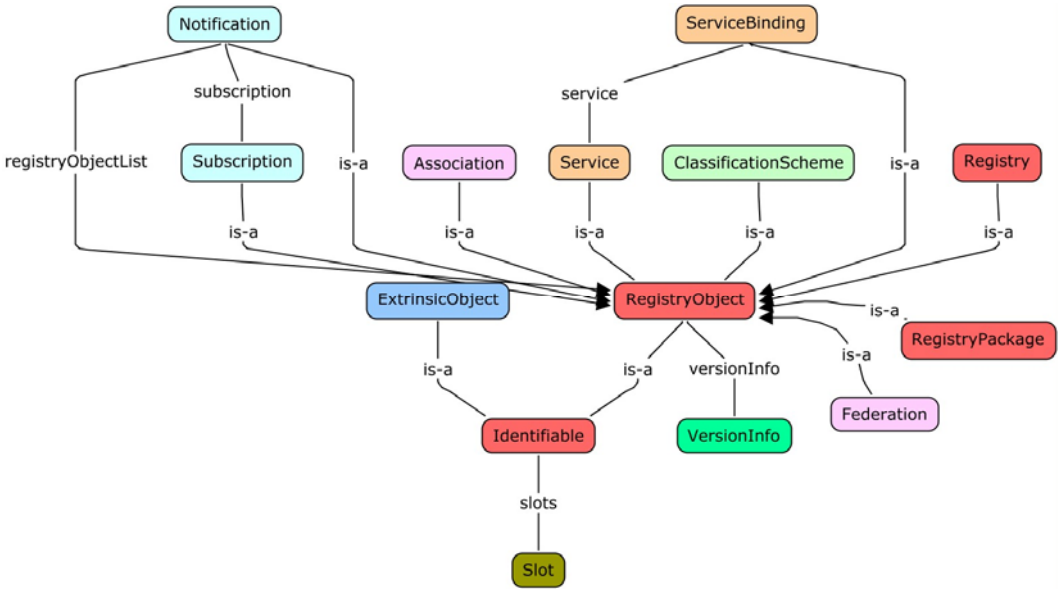


Figure 5 - CCSDS Registry Conceptual Map – Key Classes

4.2.2.2 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 classes. A data dictionary is also written as a companion to the logical model.

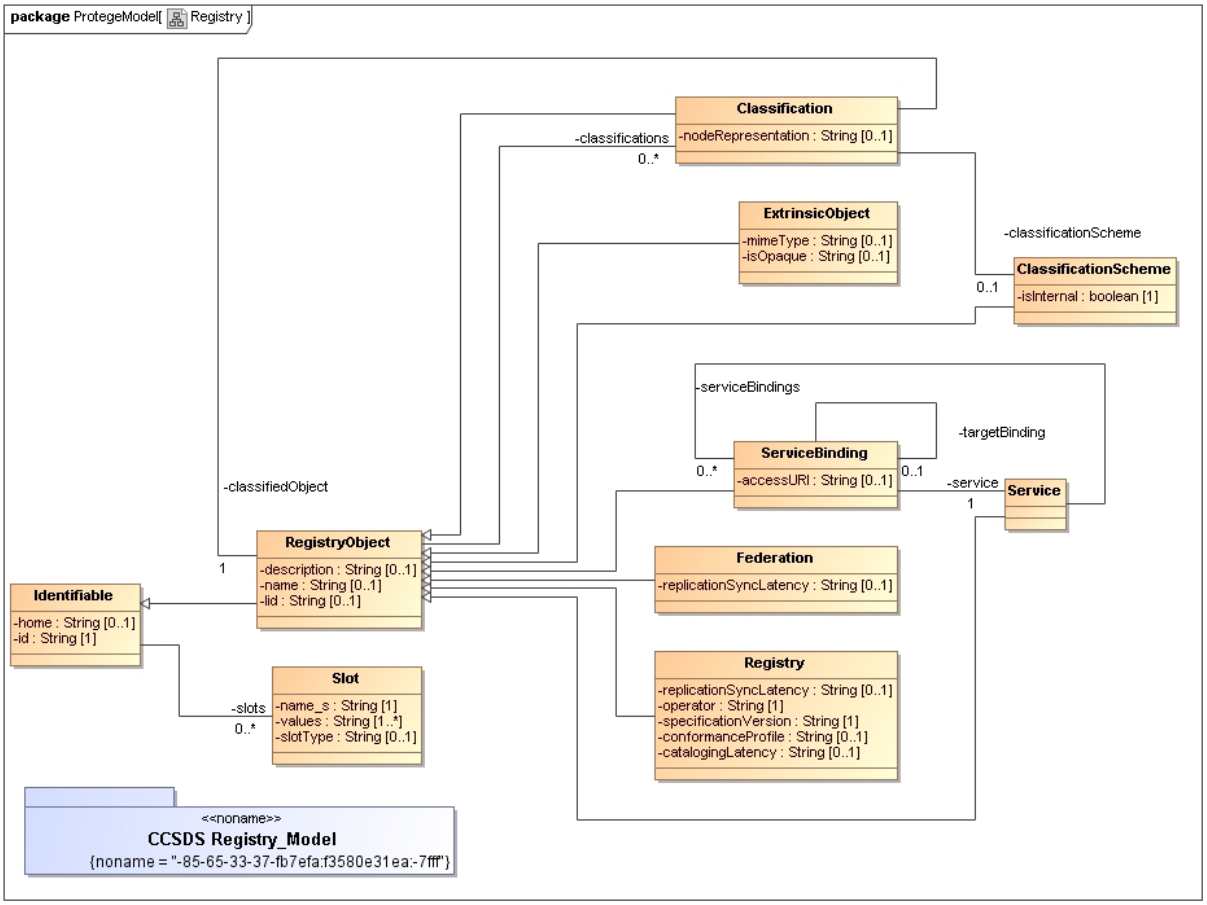


Figure 6 – Key Registry Class Definitions

5 FEDERATION

5.1 OVERVIEW

A federated registry 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 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 (i.e., provide a seamless service across different registries in different domains).

- Linking – linking content and the associated metadata in any cooperating registry (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 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 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. A federation implies a loosely coupled system where participants can join in and leave the federation without breaking the federation. It also implies that participants can function on their own when they are not a part of the federation. This represents the reality where member organizations act as independent “businesses”. The federation concept fits within these needs.

A federated architecture enables the individual cooperating organizations to function as a single federated system. The Federation class in the information model allows the creation of a Federation. A Federation is a registry object and is registered and managed as any other registry 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.

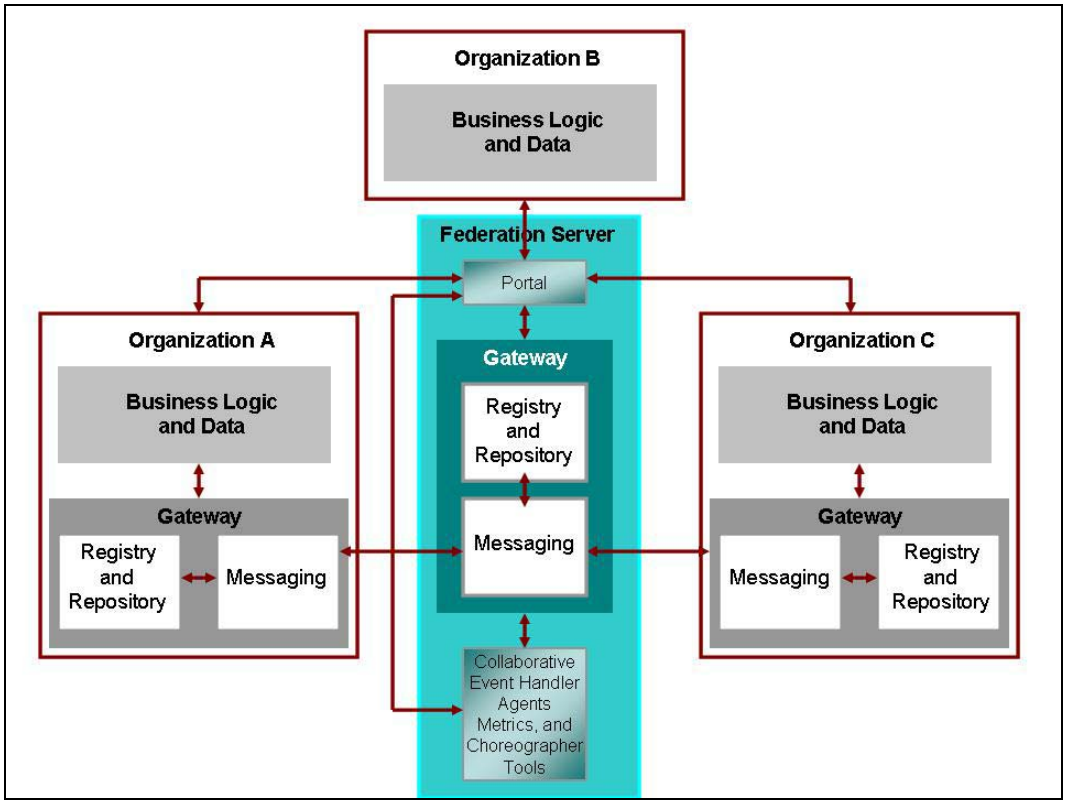


Figure 7. The Federated Reference Architecture

5.4 FEDERATED REGISTRY 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 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.

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 that has the union of all data within all the physical registries.

3. Local Caching of Data from Another Registry - A destination registry wishes to cache some or all the data of another source registry that is willing to share its data. The shared dataset is copied from the source registry to the destination registry and is visible to queries on the destination registry even when the source registry 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 is associated with a RegistryObject in another registry, and the first registry caches the second RegistryObject locally.

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

5.4.2 REGISTRY FEDERATION

A registry federation 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 federations appear as a single logical registry to registry clients.

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

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

5.4.3 QUERIES

A federation appears to registry clients as a single unified logical registry. 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 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 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 MAY be a member of multiple federations. In such cases if the federated attribute of AdhocQueryRequest has the value of true then the registry 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 in a stylized manner. Federation lifecycle operations are privileged operations. A registry SHOULD Restrict Federation lifecycle operations to registry User's that have the RegistryAdministrator role.

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

- Each registry SHOULD have exactly one Registry instance within that registry for which it is a home. The Registry instance is owned by the

RegistryOperator and may be placed in the registry using any operator specific means. The Registry instance SHOULD never change its home registry.

- A registry 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 instance as targetObject, using an associationType of HasFederationMember. The home registry 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 using SubmitObjectsRequest.
- The registry 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 leaves a federation:

- A registry 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 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 is a type of registry 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 and therefore must be described by means of additional attributes. Since the registry 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 Registry Object. 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. In some situations, a Submitting Organization may submit content that is encrypted and not even readable by the Registry. The “mime Type” attribute provides information about the type of object since the object Type is user defined and not predefined in the registry.

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.

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.).

Figure 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. See opaque attribute below.

The XML Schema extrinsic object is not opaque, and therefore allows the registry to read and process the content. Content processing, such as decomposing the XML Schema and registering each component requires an augmentation to the registry'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. The following table 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 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 XML Schema class inherits the following RegistryObject attributes and is managed according to the registry 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. – For all XML SChemas, the value will be true. This implies that the registry 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 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.

- 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. The following table 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 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 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. – For all CIOs, the value will be false. This implies that the registry will not care about the content of the CIO and the information system will be required to retrieve a CIO from the registry 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 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 API

7.1 OVERVIEW

A registry allows organizations to publish and discover Web services. Currently, two registry standards dominate: UDDI (Universal Description, Discovery, and Integration) and ebXML. With either of these, science organizations can publish a set of Web services so their internal or external participating organizations can discover them. However, integrating Web services' discovery and registration regardless of the supported registry standard can prove challenging. For example, suppose some of the organizations published their services in a UDDI registry, while others published in an ebXML registry. How does an application discover those services published by all the participating organizations?

JAXR, the Java API for XML Registries, provides a standard API for publication and discovery of Web services through underlying registries.

7.2 JAXR OVERVIEW

JAXR does not define a new registry standard. Instead, this standard Java API performs registry operations over a diverse set of registries and defines a unified information model for describing registry contents. Regardless of the registry provider, applications use common APIs and a common information model. The JAXR specification defines a general-purpose API, allowing any JAXR client to access and interoperate with any business registry accessible via a JAXR provider. In this sense, JAXR provides a Write Once, Run Anywhere API for registry operations, simplifying Web services development, integration, and portability.

7.2.1 JAXR ARCHITECTURE

The JAXR architecture defines three important architectural roles:

- A **registry provider** implements an existing registry standard, such as the OASIS (Organization for the Advancement of Structured Information) / ebXML Registry Services Specification.
- A **JAXR provider** offers an implementation of the JAXR specification approved by the Java Community Process (JCP) in May 2002. Most organizations elect to implement a JAXR provider as a facade around an existing registry provider, such as a UDDI or ebXML registry provider. Currently, the JAXR reference implementation 1.0 offers a JAXR UDDI provider implementation. A group of developers are developing an open source JAXR ebXML provider implementation at www.sourceforge.com.
- A **JAXR client** is a Java program that uses JAXR to access the registry provider via a JAXR provider. A JAXR client can be either a standalone J2SE (Java 2 Platform, Standard Edition) application or J2EE components,

such as EJBs (Enterprise JavaBeans), Java Servlets, or JSPs (JavaServer Pages). The JAXR reference implementation also supplies one form of a JAXR client, a Swing-based registry browser application.

Figure 10 illustrates how diverse JAXR clients interoperate with diverse registries using JAXR. Architecturally, JAXR clients use the API to perform registry operations, while JAXR providers implement the API. Because JAXR offers a standard API for accessing diverse registry providers and a unified information model to describe registry contents, JAXR clients, whether HTML browsers, J2EE components, or standalone J2SE applications, can uniformly perform registry operations over various registry providers.

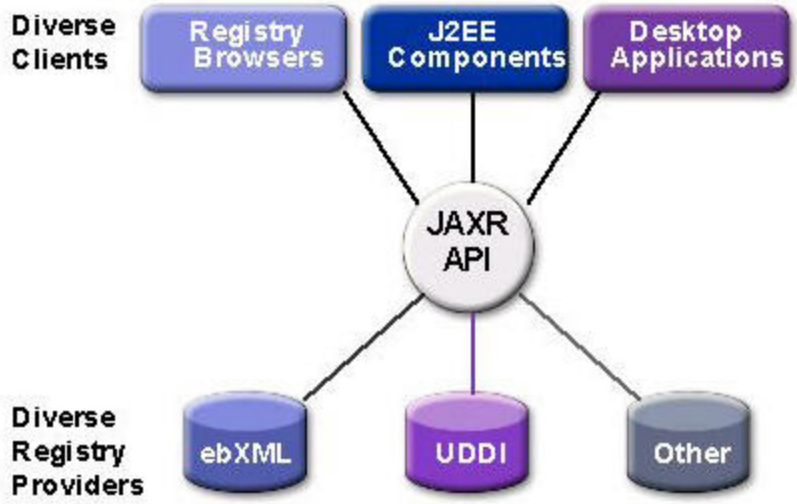


Figure 10 **JAXR interoperability with any client to any registry.**
Source: Sun Microsystems

Figure 11 shows a high-level view of the JAXR architecture. The JAXR provider shown is a JAXR pluggable provider with underlying implementations of a UDDI-specific JAXR provider and an ebXML-specific provider. The JAXR provider exposes capability-specific methods to the JAXR client via the RegistryService interface. The JAXR client queries the RegistryService and discovers the provider capability level via the CapabilityProfile interface.

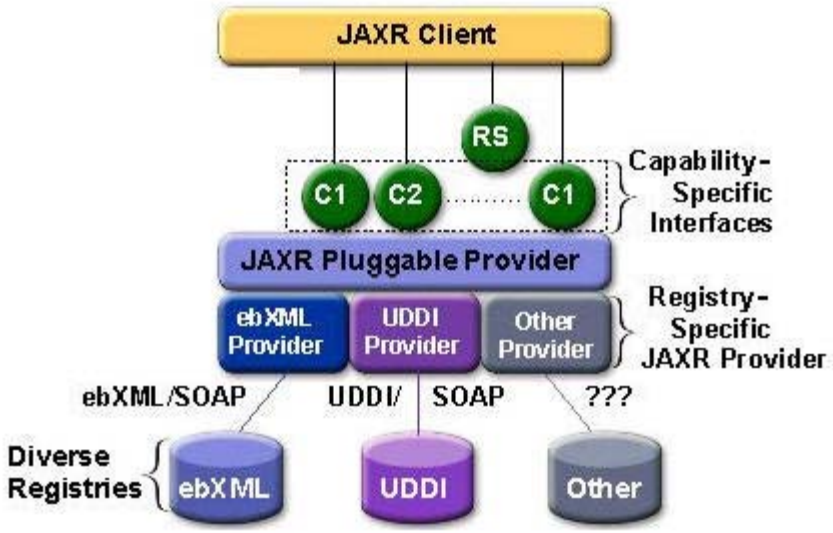


Figure 11. **JAXR architecture** Source: Sun Microsystems

Before a JAXR client can invoke capability-level methods on the JAXR provider, it must connect to the provider. First the client obtains a ConnectionFactory instance using the static method ConnectionFactory.newInstance(). The ConnectionFactory interface lets the client create the Connection using its createConnection() method. Note that the JAXR client connects with the JAXR provider, not the registry provider. The JAXR provider acts as a proxy on the client's behalf, directing and invoking methods on the appropriate registry provider. The connection maintains client state. In addition, the JAXR client dynamically sets its authentication information and communication preference on the connection any time during the connection's lifetime. Please refer to the code demonstrating how a JAXR client connects to a JAXR provider in the development examples later in the article.

After the JAXR client invokes JAXR capability-level methods, the JAXR provider transforms these methods into registry-specific methods and executes requests to the underlying registry providers. After the registry providers process the requests and return registry-specific results to the JAXR provider, the JAXR provider transforms the information into JAXR information model RegistryObjects and returns them to the JAXR client. The RegistryObject interface is an abstract interface that provides the common information such as key, name, and description for the more specialized JAXR information model interfaces. Note that the communication protocol between a JAXR provider and a registry provider is registry provider-specific and transparent to the JAXR client. For example, a JAXR provider communicates with the UDDI registry provider by exchanging basic SOAP messages, while the JAXR provider communicates with the ebXML registry provider through SOAP messaging or ebXML message service.

7.2.1.1 Capability Profiles

Because some diversity exists among registry provider capabilities, the JAXR expert group decided to provide multilayer API abstractions through *capability profiles*. Each method of a JAXR interface is assigned a capability level, and those JAXR methods with the same capability level define the JAXR provider capability profile.

Currently, JAXR defines only two capability profiles: 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 JAXR providers must implement a level 0 profile. A JAXR client application using only those methods of the level 0 profile can access any JAXR provider in a portable manner. JAXR providers for UDDI must be level 0 compliant.

JAXR providers can optionally support the level 1 profile. The methods assigned to this profile provide more advanced registry capabilities needed by more demanding JAXR clients. Support for the level 1 profile also implies full support for the level 0 profile. JAXR providers for ebXML must be level 1 compliant. A JAXR client can discover the capability level of a JAXR provider by invoking methods on the CapabilityProfile interface. If the client attempts to invoke capability level methods unsupported by the JAXR provider, the provider will throw an UnsupportedCapabilityException.

7.2.1.2 Registry Service Interfaces

As mentioned in the previous section, the JAXR provider supports capability profiles that group the methods on JAXR interfaces by capability level. RegistryService exposes the JAXR provider's key interfaces, that is, Web services discovery and registration. The JAXR client can obtain an instance of the RegistryService interface by invoking the getRegistryService() method on the connection established between the JAXR client and JAXR provider. Once the JAXR client has the RegistryService, it can obtain the primary registry interfaces and perform life-cycle management and query management through the JAXR provider.

The JAXR specification defines two life-cycle management interfaces:

- BusinessLifeCycleManager for level 0
- LifeCycleManager for level 1

BusinessLifeCycleManager defines a simple business-level API for life-cycle management. This interface resembles the publisher's API in UDDI, which should prove familiar to the UDDI developer. For its part, LifeCycleManager interface provides complete support for all life-cycle management needs using a generic API.

Life-cycle management includes creating, saving, updating, deprecating, and deleting registry objects. In addition, the LifeCycleManager provides several factory methods to create JAXR information model objects. In general, life-cycle management operations are privileged, while a user can use query management operations for browsing the registry.

JAXR's top-level interface for query management, QueryManager, has two extensions:

- BusinessQueryManager for level 0
- DeclarativeQueryManager for level 1

Query management deals with querying the registry for registry data. A simple business-level API, the BusinessQueryManager interface provides the ability to query for the most important high-level interfaces in the information model, such as Organizations, Services, ServiceBindings, ClassificationSchemes, and Concepts. Alternatively, the DeclarativeQueryManager interface provides a more flexible, generic API, enabling the JAXR client to perform ad hoc queries using a declarative query language syntax. Currently, the only declarative syntaxes JAXR supports are SQL-92 and OASIS/ebXML Registry Filter Queries. As noted in the JAXR specification, ebXML registry providers optionally support SQL queries. If a registry provider does support SQL queries, the JAXR ebXML provider will throw an UnsupportedCapabilityException on DeclarativeQueryManager methods.

7.2.1.3 JAXR Information Model

Invoking life-cycle and query management methods on the JAXR provider requires the JAXR client to create and use the JAXR information model objects. The JAXR information model resembles the one defined in the ebXML Registry Information Model 2.0, but also accommodates the data types defined in the UDDI Data Structure Specification. Although developers familiar with the UDDI information model might face a slight learning curve, once understood, the JAXR information model will provide a more intuitive and natural interface to most developers.

Most JAXR information-model interfaces are derived from the abstract RegistryObject interface, which defines the common state information, called attributes, that all registry objects share. Example attributes include key, name, and description. The InternationalString interface defines attributes that must be

internationalization compatible, such as name and description. The InternationalString interface contains a collection of LocalizedStrings, where each LocalizedString defines locale, character set, and string content.

The RegistryObject interface also defines collections of Classifications, ExternalIdentifiers, ExternalLinks, and Associations. The BusinessQueryManager often uses those collections as parameters in its find methods.

Also specializations of the RegistryObject interface, the concrete interfaces Organization, Service, ServiceBinding, Concept, and ClassificationScheme provide additional state information. For example, the Organization interface defines a collection of Services, and Service defines a collection of ServiceBindings. A ServiceBinding might contain a collection of SpecificationLinks. UDDI developers should be familiar with these concrete interfaces; they map quite well to the five major UDDI data types shown in the table below.

7.3 JAXR API SUMMARY

JAXR, the Java API for XML Registries, provides a standard API for publication and discovery of Web services through underlying registries. The following table presents a summarized list of the individual API's together with a brief description. APIs with similar functions but different arguments were combined. For example, the two addAssociation APIs, one for a single association and the other for collection of associations were combined. A table in Annex 4 provides the mapping for the JAXR API's to the use cases of Chapter 4 and to the CCSDS XML/Schema tool APIs.

API	Description
addRegistryObjects	Adds RegistryObjects.
getRegistryObjects	Gets the collection of member RegistryObjects of this RegistryPackage.
removeRegistryObjects	Removes RegistryObject.
addAssociation	Adds specified Association for this object.
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.
getAssociations	Gets all Associations where this object is the source.
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.
getSubmittingOrganization	Gets the Organization that submitted this RegistryObject.
removeAssociation	Removes specified Association from this object.
removeClassification	Removes specified Classification from this object.
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.
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.
setName	Sets user-friendly name of object in repository.
toXML	Returns a registry provider specific XML representation of this Object.

7.4 SPECIALIZED FACADE INTERFACES

7.4.1 XML SCHEMAS

The following table lists the APIs for the FreebXML Façade implemented for the XML Schema Tool. [JavaDoc descriptions will be inserted when available.]

API	Description
addSchemaAssociatedObject	?
approveSchema	?
createObjKey	?
deleteAssociationsForObject	?
deleteObject	?
deleteSchema	?
findAllExtrinsicObjects	?
findLatestVersionKey	?
getAllSchemaVersions	?
getObjectContentIfExists	?
getPackage	?
getRepositoryItem	?
getRepositoryItemForLatestVersion	?
getSchema	?
getSchemaAssociatedObjects	?
getSchemaImportsIncludes	?
getSchemaRepositoryObects	?
ingestSchema	?
publishTargetNamespace	?
updateSchema	?
Validate	?

8 LIFECYCLE MANAGEMENT

8.1 OVERVIEW

This section defines the protocols supported by 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].

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 epository 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 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

A1.1 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

A1.2 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).

A1.2.1.1 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

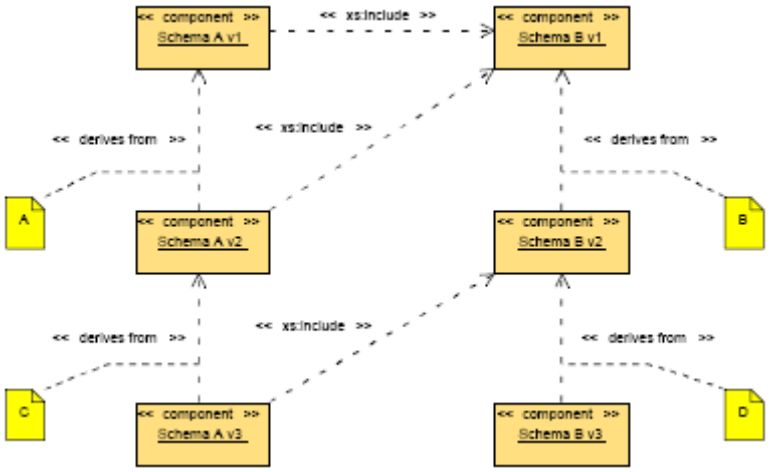
	in particular install the repository; manage security level and user authentication/rights. An administrator may be seen as a "Super User".
Application	Any software interacting with the XML Schema Repository. In the current definition it is not foreseen that Application would be granted to perform administration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

A1.2.1.2 XML Schema Submission Use Cases

Use Case Amalfi-1

A1.2.1.3 Versioning Use Cases

Use Case Amalfi-2

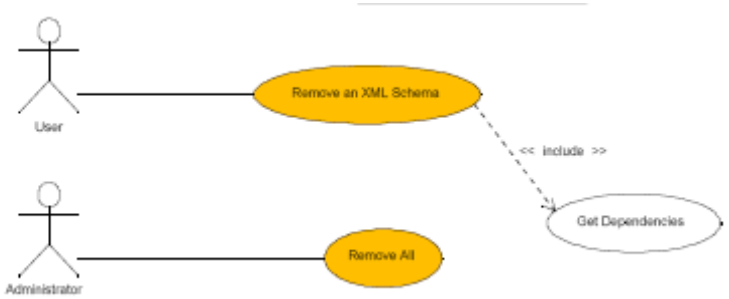


A1.2.1.4 XML Schema Retrieval Use Cases

Use Case Amalfi-3

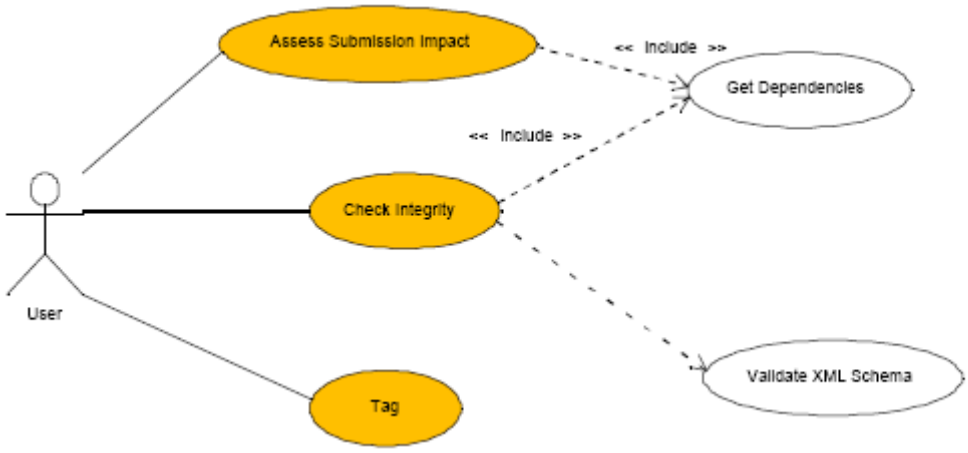
A1.2.1.5 XML Schema Removal Use Cases

Use Case Amalfi-4



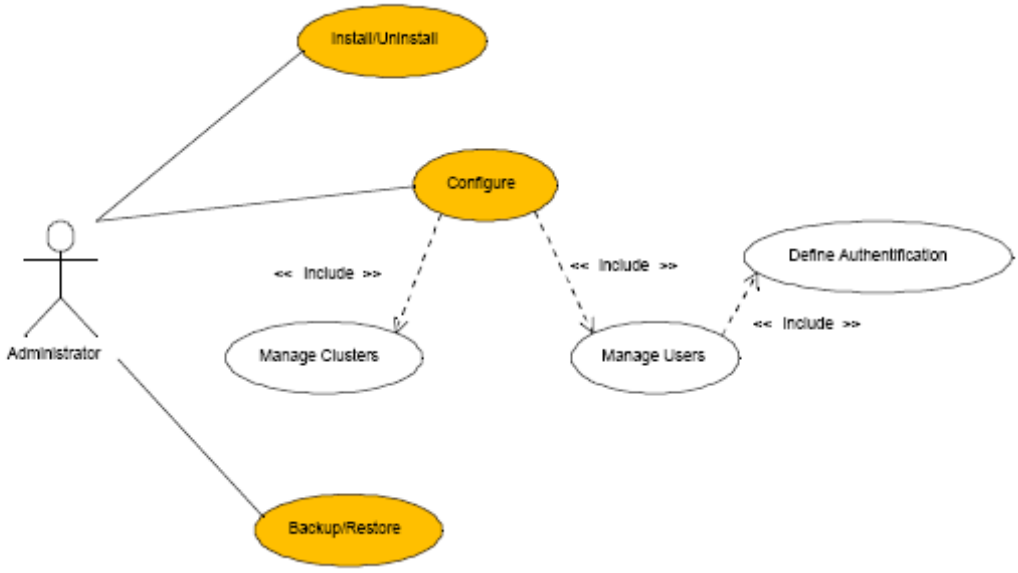
A1.2.1.6 Revision Control Use Cases

Use Case Amalfi-5



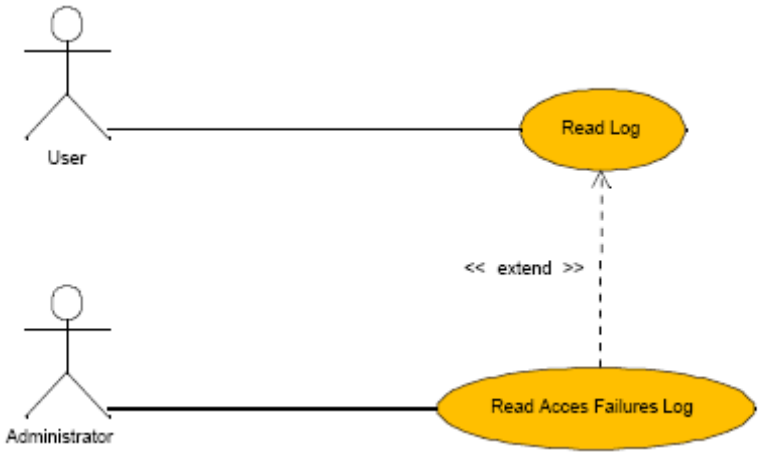
A1.2.1.7 Repository Administration Use Cases

Use Case Amalfi-6



A1.2.1.8 Logging/Monitoring Use Cases

Use Case Amalfi-7



A1.3 JPL DEEP SPACE NETWORK INFORMATION SERVICE ARCHITECTURE REGISTRY USE CASES

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

A1.3.1.1 Background

The Deep Space Network Information Service 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.

A1.3.1.2 Use Cases

A1.3.1.3 General Use Cases

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

Migrate Registry to Operations	

A1.3.1.4 Blanket Registry Use Cases

Backup Registry	Restore Registry
Provide Alternate Registry	

A1.3.1.5 Data Element Use Cases

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

A1.3.1.6 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	

A1.3.1.7 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	

A1.3.1.8 Namespace/Domain Registry Use Cases

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

A1.3.1.9 Service Registry Use Cases

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

A1.4 CCSDS SERVICE LINK EXCHANGE (SLE) WG

A1.4.1.1 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.

A1.4.1.2 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

A1.5 CCSDS NAVIGATION WG

A1.5.1.1 Background

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

A1.5.1.2 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

A1.6 CCSDS MISSION OPERATIONS SERVICES

A1.7 COMMON SM&C USE CASES

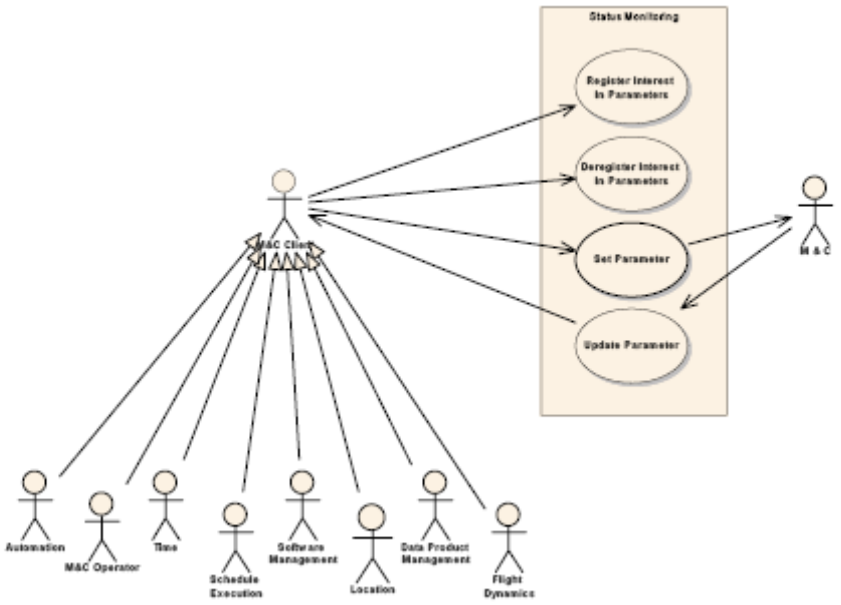
Use Case MOS-1	Register Interest
Brief description	Use case allows a user to register to receive updates It is espected that the registering of interest would involve the sending of some kind of filter and would also require some kind of privilege. The requet may also specify that only the current state 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

A1.8 CORE SM&C USE CASES

A1.8.1.1 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

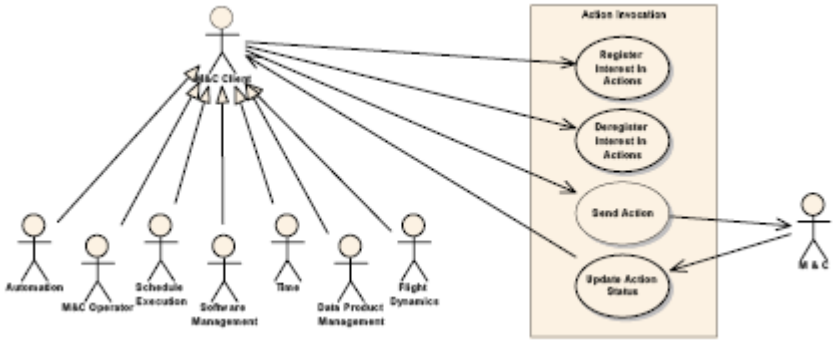
	parameter state and subsequent changes in the state should be supplied (continuous).
Primary Actor	M&C Client
Preconditions	The M&C Subsystem which provides the Parameter values must be available.
Main Success Scenarios	• All the parameter references provided by the M&C Status Client are validated. • 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

A1.8.1.2 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.

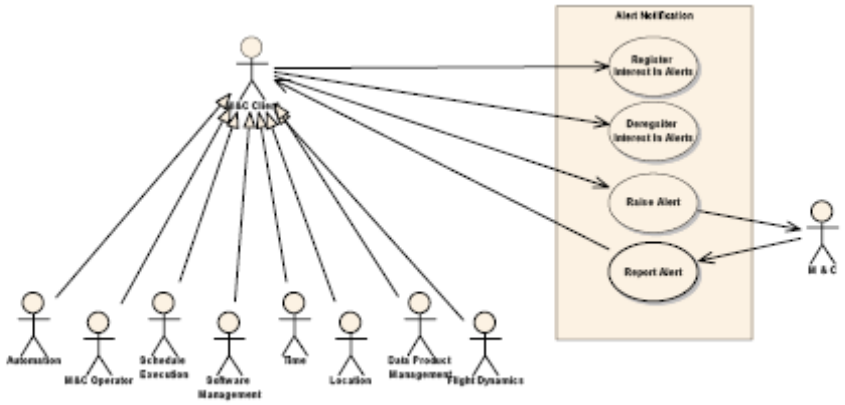
Use Case MOS- 6	**Update Action Status**
Brief Description	Use case allows a M&C System to report an update in the status of an Action.
Primary Actor	M&C (Subsystem)
Preconditions	The action must have been sent by a M&C Client.
Main Success Scenarios	• The action is validated. • The action status is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

A1.8.1.3 Alert Notification



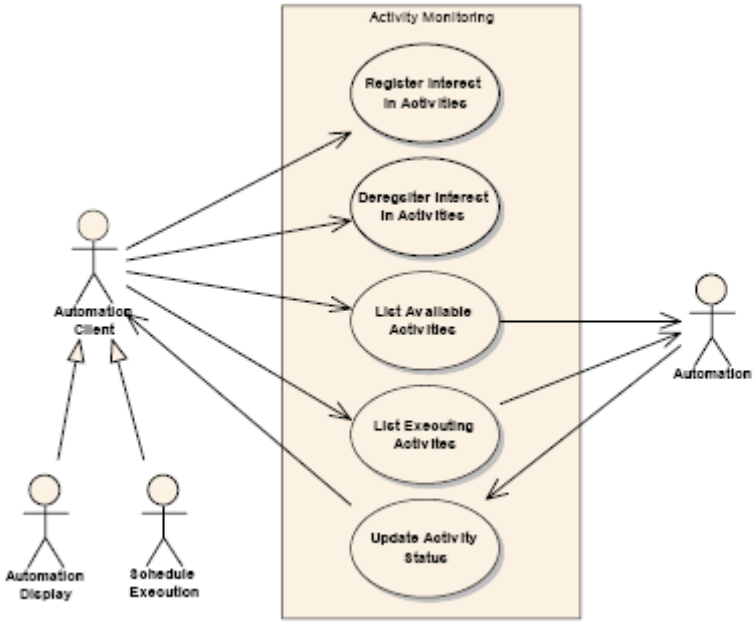
Use Case MOS-9	Register Interest In Alerts
Brief Description	Register to receive notification of Alerts. The Alerts for which notifications are required can be specified by any of the following methods : Providing the definition identifiers – of the Alerts for which updates are required. Providing the 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.

A1.9 OPERATIONS AUTOMATION USE CASES

A1.9.1.1 Activity Control Use Cases

A1.9.1.2 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

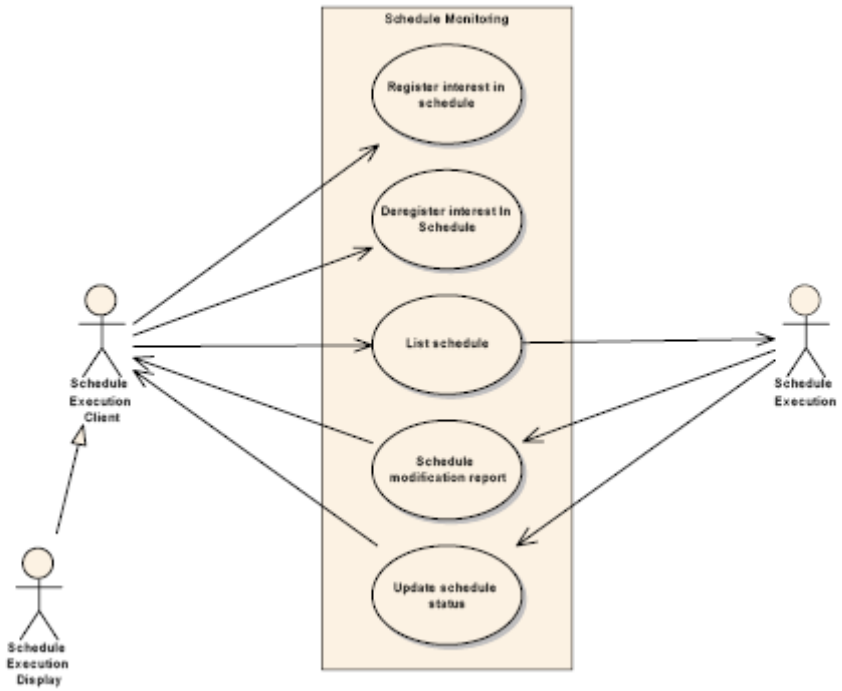
A1.10 OPERATIONS SCHEDULING USE CASES

A1.10.1.1 Schedule Level Control Use Cases

A1.10.1.2 Schedule Maintenance

A1.10.1.3 Schedule Activity Level Control Use Cases

A1.10.1.4 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.

A1.11 OPERATIONS PLANNING USE CASES

A1.11.1.1 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.

A1.11.1.2 Plan Level Maintenance Use Cases

A1.11.1.3 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.

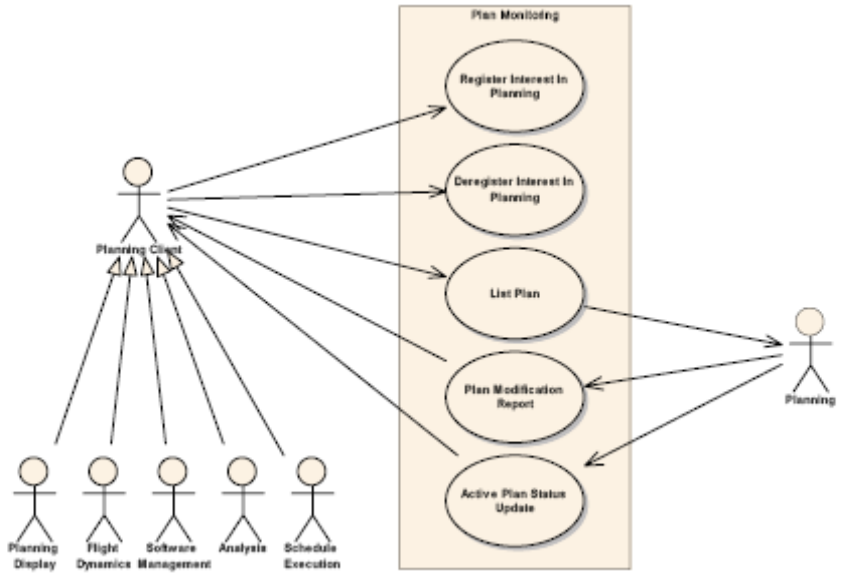
Use Case MOS-30	**Modify Constraint**
Brief Description	Modify an existing constraint.
Primary Actor	Planning Client
Preconditions	Constraint exists.
Main Success Scenarios	• Constraint is modified.

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.

A1.11.1.4 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.

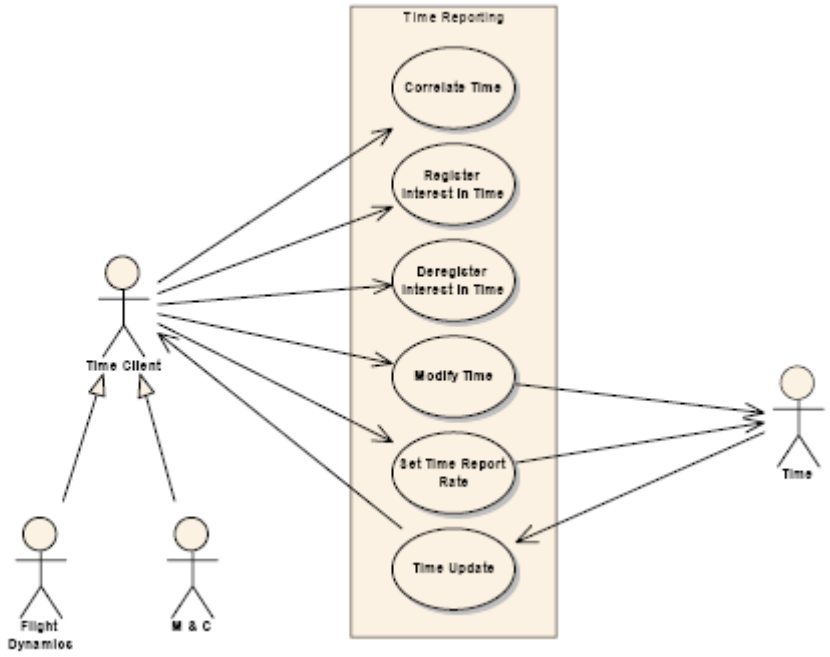
Use Case MOS-36	**Plan Modification Report**
Brief Description	Summary report of any modification made to the active plan.
Primary Actor	Planning
Preconditions	Client had registered for updates.
Main Success Scenarios	• All registered clients receive the update.

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.

A1.12 GUIDANCE, TRACKING AND SYNCHRONISATION USE CASES

A1.12.1.1 Time use cases



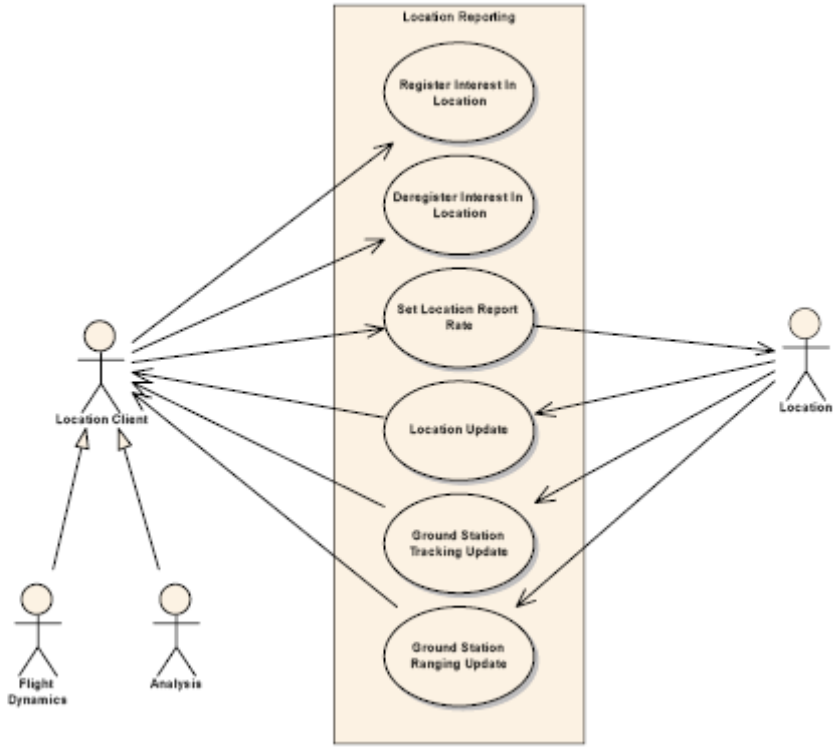
Use Case MOS-38	**Register Interest In Time**
Brief Description	Allows a client to register interest in time reports.
Primary Actor	Time Client
Preconditions	None.

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.

A1.12.1.2 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.

A1.12.1.3 Location Control use cases

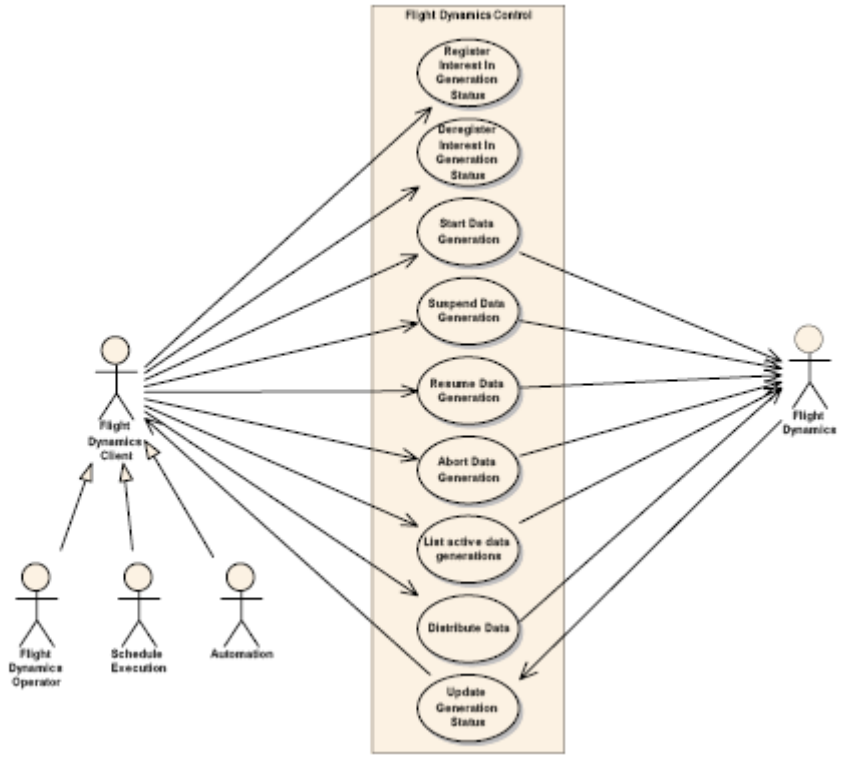
Use Case MOS-46	**Register Interest In Location Control**
Brief Description	Allows a client to register interest in location control updates.
Primary Actor	Location Client
Preconditions	None.
Main Success Scenarios	• Client is registered for location control updates.

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.

A1.12.1.4 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.

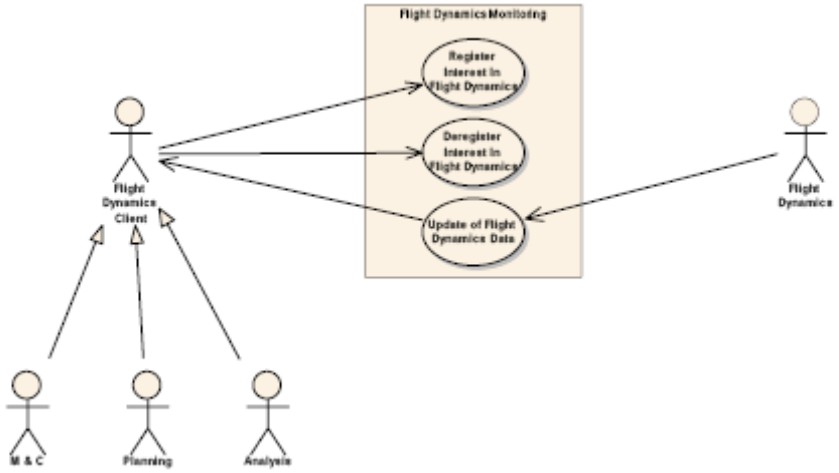
Use Case MOS-51	Deregister Interest In Generation Status
Brief Description	Allows a previously registered client to stop receiving flight dynamics generation status.

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.

A1.12.1.5 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

A1.13 REMOTE SOFTWARE MANAGEMENT USE CASES

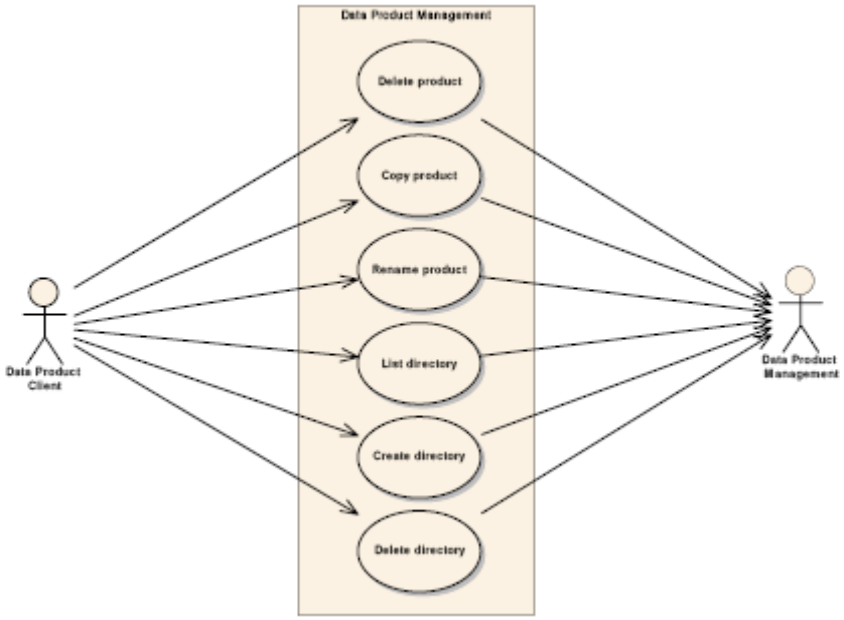
Use Case MOS-57	Register Interest In Software Management
Brief Description	Allows a client to register interest in Software Management data.
Primary Actor	Software Management Client
Preconditions	None.
Main Success Scenarios	• Client is registered for data notification.

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.

A1.14 PAYLOAD DATA PRODUCT MANAGEMENT USE CASES

A1.14.1.1 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.

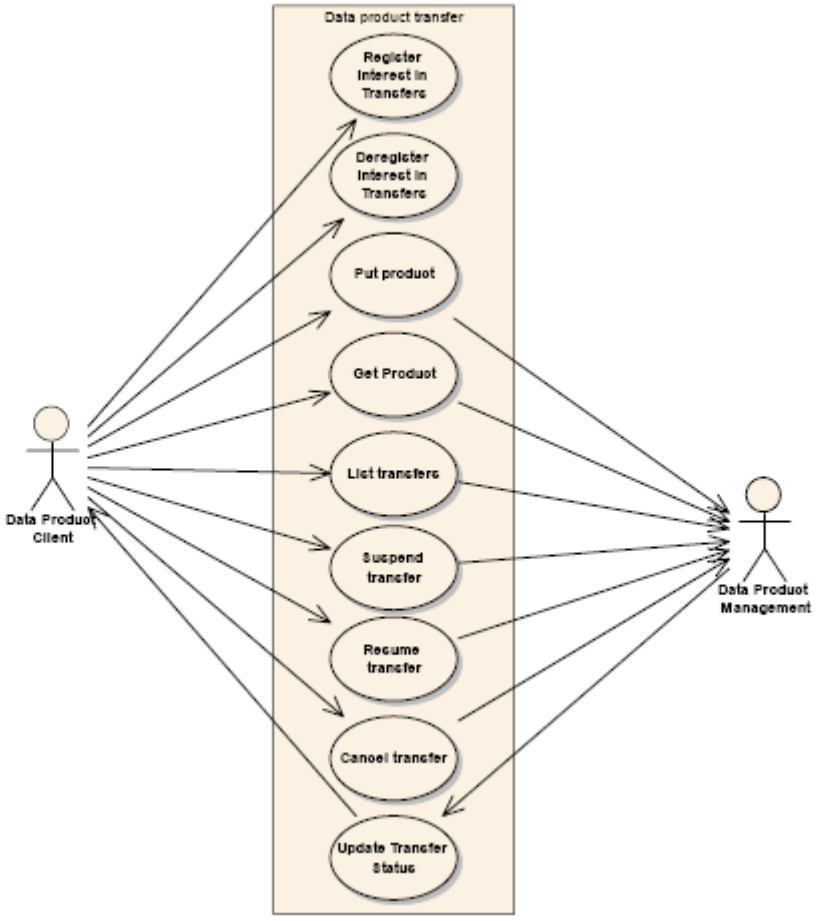
Use Case MOS-63	List Directory
Brief Description	Returns a list of files in the specified directory.

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.

A1.14.1.2 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.

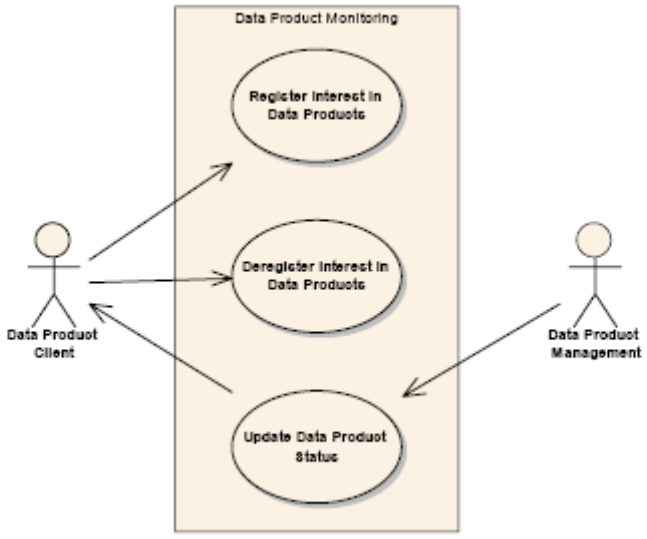
Use Case MOS-67	**Deregister Interest In Transfers**

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.

A1.14.1.3 Data Product Monitoring use cases

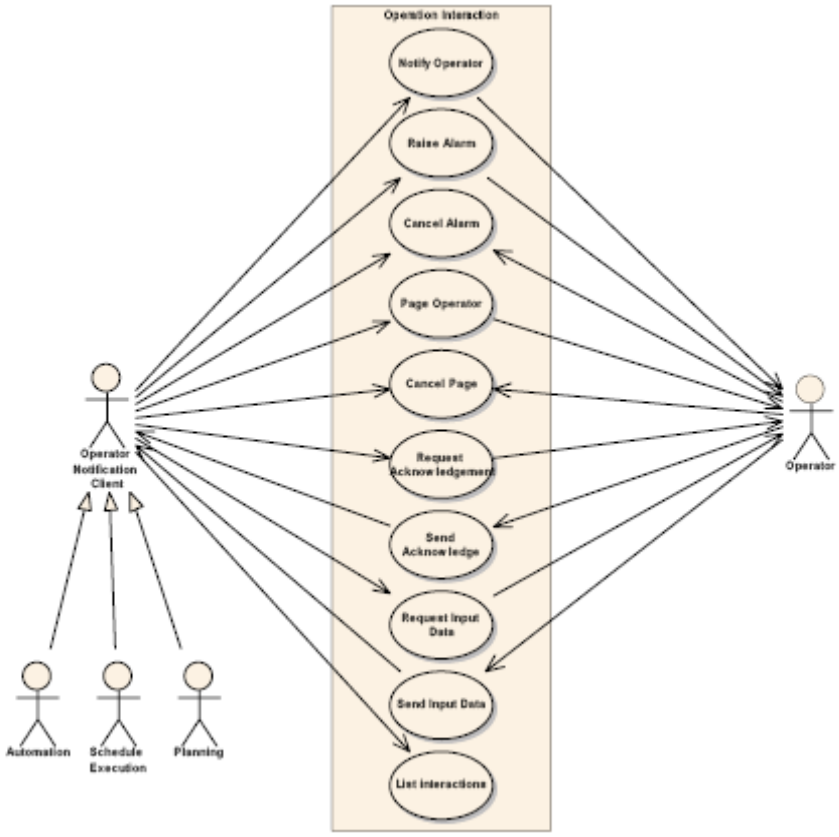


Use Case MOS-70	Register Interest In Data Products
Brief Description	Allows a client to register interest in data product status updates.
Primary Actor	Data Product Client
Preconditions	None.
Main Success Scenarios	• Client is registered for data product notification.

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.

A1.15 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's 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, getRepositoryItemForLatestVer 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