Planetary Data System

Registry Service

Software Requirements and Design Document (SRD/SDD)

Sean Hardman
Paul Ramirez

September 28, 2010
Version 0.6

Jet Propulsion Laboratory
Pasadena, California

CHANGE LOG

Revision	Date	Description	Author
0.1	2009-12-09	Initial draft.	S. Hardman, P. Ramirez
0.2	2010-01-25	Updated the use cases and requirements and filled out architecture section.	S. Hardman
0.3	2010-03-06	Incorporated comments from the System Design Working Group and added information on the REST-based interface and prototype analysis.	S. Hardman
0.4	2010-05-13	Updated reference documents, replaced “artifact” with “product”, removed references to repository support and added some more architecture and design detail.	S. Hardman
0.5	2010-07-11	Returned to using the term “artifact” where appropriate, incorporated several comments by Mike Martin and added a use case and requirements for the undeprecate action.	S. Hardman
0.6	2010-09-28	Removed the API requirements since they are now located in the General Service requirements document. Cleaned up the remaining requirements mapping.	S. Hardman

TABLE OF CONTENTS

1.0 INTRODUCTION

The PDS 2010 effort will overhaul the PDS data architecture (e.g., data model, data structures, data dictionary, etc) and deploy a software system (online data services, distributed data catalog, etc) that fully embraces the PDS federation as an integrated system while leveraging modern information technology.

This service provides functionality for tracking, auditing, locating, and maintaining artifacts within the system. These artifacts can range from data files and label files, schemas, dictionary definitions for objects and elements, services, etc.

1.1 Document Scope and Purpose

This document addresses the use cases, requirements and software design of the Registry service within the PDS 2010 data system. This document is intended for the reviewer of the service as well as the developer and tester of the service.

1.2 Method

This combined Software Requirements and Software Design Document (SRD/SDD) represents the software by defining use cases and requirements and by using architecture diagrams, functional descriptions, context diagrams and data flow diagrams for the high-level design. UML diagrams will illustrate the detailed design.

1.3 Notation

The numbering of the requirements in this document will be formatted as **LX.REG.AA.X**, where:

- **LX** represents the requirements level where X is a number.
- **REG** is an abbreviation representing the registry requirements section for the specified level.
- **AA** is a two-letter abbreviation representing the requirement sub-category (optional).
- **X** is a unique number within the section and optional sub-category for the requirement.

Following the text of a requirement may be a reference to the requirement or use case from which it was derived. The reference will be in parenthesis. A paragraph following a requirement, which is indented and has a reduced font size, represents a comment providing additional insight for the requirement that it follows. This comment is not part of the requirement for development or testing purposes.

1.4 Controlling Documents

[1] Planetary Data System (PDS) Level 1, 2 and 3 Requirements, March 26, 2010.

[2] Planetary Data System (PDS) 2010 Project Plan, February 2010.

[3] Planetary Data System (PDS) 2010 System Architecture Specification, Version 1.1, May 5, 2010.

[4] Planetary Data System (PDS) 2010 Operations Concept, February 2010.

[5] Planetary Data System (PDS) General System Software Requirements Document (SRD), Version 0.1, September 27, 2010.

1.5 Applicable Documents

[6] CCSDS Registry and Repository Reference Model, February 2, 2010.

[7] PDS4 Information Model Specification, PDS4 Information Model Specification Team.

[8] Planetary Data System Search Service Software Requirements and Design Document (SRD/SDD), Version 0.2, July 12, 2010.

[9] Registry Services, May 31, 2009.

1.6 Document Maintenance

The component design will evolve over time and this document should reflect that evolution. This document is limited to design content because the specification content will be captured in separate documentation (e.g., Installation Guide, Operation Guide, etc.). This document is under configuration control.

2.0 COMPONENT DESCRIPTION

The Registry service provides the track and locate artifact function for the PDS 2010 system (referred to as the "system" from this point forward). The intent of this service is to facilitate tracking, auditing and maintenance of artifacts within PDS (e.g., data, dictionary definitions, schemas, services, etc.). The following diagram details the context of the Registry service, represented as the Inventory, Dictionary, Document and Service services, within the system:

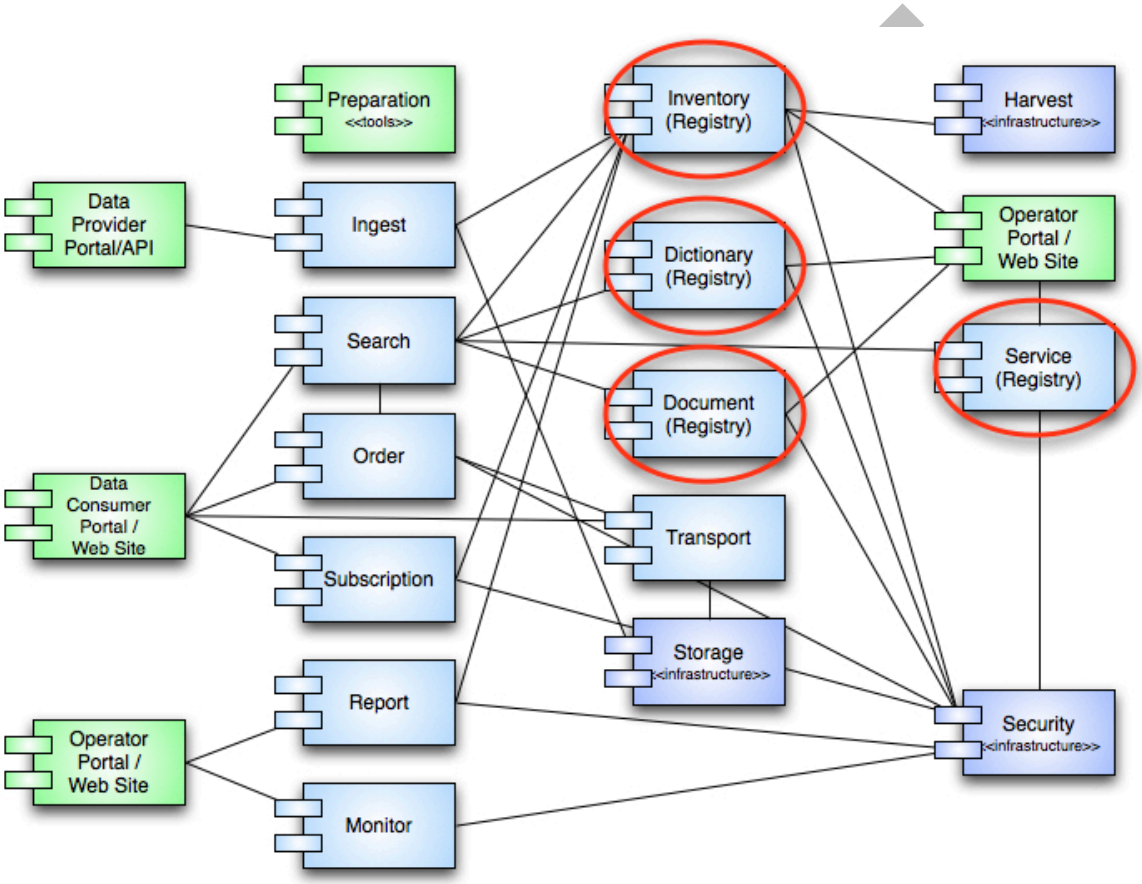

Figure 1: Registry Service Context

Within the system, the Registry service will have a limited set of external interfaces and will mostly interact with other system components. The rationale behind this is to reduce the complexity of the service as its functions are at the core of the system. Other services will build upon the information maintained in any given registry and will expose this registry-based information via external interfaces. This separation of concerns will help the system evolve as any external requirements can be leveraged on other services and thus reducing the impact to this core component.

As depicted in the diagram above, the Registry service supports several interfaces to other services in the system. In general, these services will interact

with the registry to inform the service about a new managed artifact or lookup/update basic information about an existing registered artifact. The registry will maintain three types of registrations:

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

Digital Object Entry
This type of entry tracks back to a physical set of bits. In the current PDS infrastructure, this would be items such as products consisting of a label and data files. In the proposed system, this expands to include any item of interest (e.g., documents, schemas, etc.).

Relationship Entry
This type of entry will serve as a means to tie registered products together. Such support is necessary for example to correlate collections to the set of products contained within. These relationships may span registries and thus the need for coordination amongst registries exists. Example product relationships include associations with an investigation product, an instrument product and a target product. The supported list of relationships can be found in the Information Model [7].

Although the current PDS system does not have an official registry service, there are pieces within the existing architecture that act in the capacity of a registry. One example of this is the current catalog, which maintains data set, mission, instrument, and other descriptions. Yet, another example is the archive directory structure itself, which organizes and associates data and label files for a particular data set or volume. Moreover, nodes generally have a catalog of products that participates in the existing infrastructure through product and/or profile servers. A Registry service instance would not seek to replace the nodes existing catalog but act as the infrastructure component equivalent. The following is an accounting of logical registries that would be available within the system:

Inventory
As indicated above this registry instance serves as a means to capture the products within the PDS. Registration of products will occur by crawling local repositories at the nodes. Products will remain within their local repositories and only enough information to locate and audit the product is gathered. This information will include, but not be limited to: access points, checksum, file name, and file size.

Dictionary
This registry captures and stores object, group and element definitions that make up the data dictionary. Management of these definitions occurs in the Information Model [7], which exports this information periodically to this logical instance of the service.

Document
With the transition to XML, management of schemas, which govern XML instance files (e.g., product labels), becomes of utmost importance. Schemas must be captured and readily available and this registry will provide this role.

Service
This registry captures descriptions about services provided by the system. PDS participants can share their services via this registry to help promote reuse. These descriptions could evolve over time from simple documentation in the form of a web page or document to something along the lines of a WSDL or WADL formatted description. The service registry will not dictate interaction with a given service but rather exist as a means to document and promote existing services.

The service defined in this document will provide the PDS 2010 system with a single implementation of registry capabilities for use by the other services and applications within the system. This service is tailor-able depending on the type of registry and types of artifacts to be registered with a given instance.

3.0 USE CASES

A use case represents a capability of the component and why the user (actor) interacts with the system. It should be at a high enough level so as not to reveal or imply the internal structure of the system. An actor is an object (e.g., person, application, etc.) outside the scope of the component but interacts with the component. This section captures the use cases for the Registry service based on the description of the service from the previous section as well as use cases defined in the CCSDS Registry and Repository Reference Model [6]. These use cases will be used in the derivation of requirements for the service. The following diagram details the use cases:

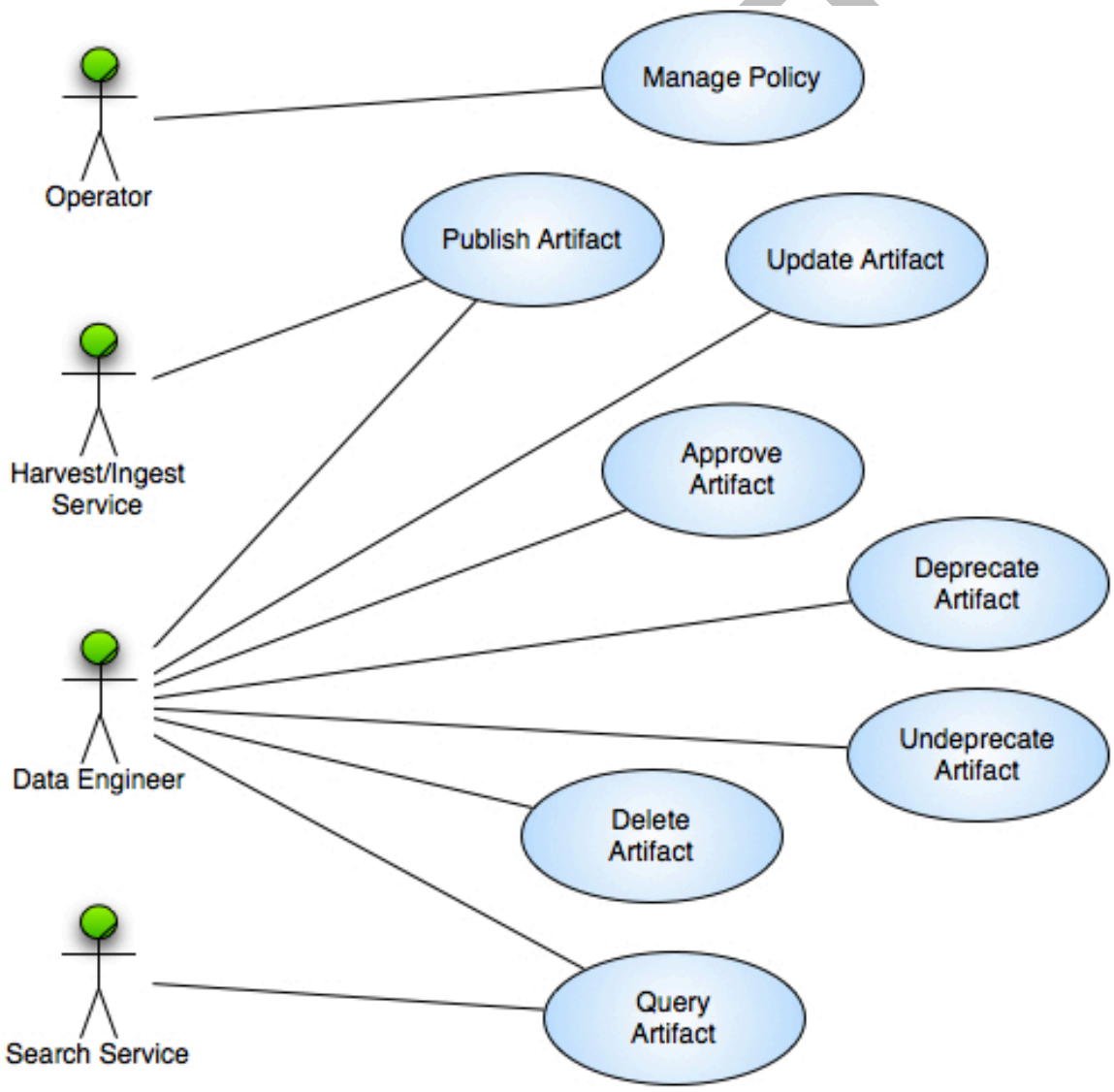

Figure 2: Registry Service Use Cases

The above diagram identifies the following actors (represented as stick figures):

Data Engineer
This actor represents a portion of the PDS Technical group that curates the data before and after it enters the PDS system.

Harvest/Ingest Service
This actor represents the software within the system that will perform automated registration of artifacts.

Operator
This actor represents a portion of the PDS Technical group that is responsible for configuring and monitoring the system.

Search Service
This actor represents the software within the system that will query for registered products.

The following sections detail the use cases identified in the above diagram.

3.1 Manage Policy

The Registry service is policy driven with regard to the types of artifacts that it registers, the associated metadata it expects to receive for an artifact and the allowed operations on a type of artifact. This use case pertains to the Operator actor.

1. Operator authenticates for access to the Registry service interface (include Security service Authenticate User use case [8]).
2. Operator submits an update to the Registry service policy to add, modify or delete a type of artifact via the Registry service interface.
3. Registry service accepts (verifies input against constraints) and commits (updates the underlying metadata store) the operation.

3.2 Publish Artifact

Register artifacts with the system for the purpose of tracking, discovery and retrieval. This use case pertains to the Ingest and Harvest services that will perform automated registration of artifacts. It also pertains to the Data Engineer who will perform ad hoc registrations of artifacts within the system.

1. Ingest/Harvest service authenticates for access to the Registry service API (include Security service Authenticate User use case [8]).
2. Ingest/Harvest service submits an artifact for registration via the Registry service API.
3. Registry service validates the metadata submitted for the artifact.

4. Registry service assigns a version to the artifact based on the PDS identifier.
5. Registry service records the metadata associated with the artifact in the underlying metadata store.

Alternative: Ad Hoc Registration
At step 1, the Data Engineer initiates the artifact registration.

a. Data Engineer authenticates for access to the Registry service interface (include Security service Authenticate User use case [8]).
b. Data Engineer submits an artifact for registration via the Registry service interface.
c. Return to primary scenario at step 3.

3.3 Update Artifact

Update a registered artifact and its associated metadata. This use case pertains to the Data Engineer who will perform artifact registration updates within the system.

1. Data Engineer authenticates for access to the Registry service interface (include Security service Authenticate User use case [8]).
2. Data Engineer submits an updated artifact for registration via the Registry service interface.
3. Registry service validates the metadata submitted for the artifact.
4. Registry service records the metadata associated with the artifact in the underlying metadata store.

3.4 Approve Artifact

Approve registered artifacts in order to make them visible to the public. This use case pertains to the Data Engineer who will approve registered artifacts.

1. Data Engineer authenticates for access to the Registry service interface (include Security service Authenticate User use case [8]).
2. Data Engineer marks a registered artifact as approved via the Registry service interface.
3. Registry service records the approval in the underlying metadata store.

3.5 Deprecate Artifact

Deprecate registered artifacts when no longer pertinent. This could be due to the availability of a newer version of the artifact. This use case pertains to the Data Engineer who will deprecate registered artifacts.

1. Data Engineer authenticates for access to the Registry service interface (include Security service Authenticate User use case [8]).
2. Data Engineer marks a registered artifact as deprecated via the Registry service interface.
3. Registry service records the deprecation in the underlying metadata store.

3.6 Undeprecate Artifact

Undeprecate registered artifacts when their pertinence has been restored. This use case pertains to the Data Engineer who will undeprecate registered artifacts.

4. Data Engineer authenticates for access to the Registry service interface (include Security service Authenticate User use case [8]).
5. Data Engineer marks a registered artifact as undeprecated via the Registry service interface.
6. Registry service records the undeprecation in the underlying metadata store.

3.7 Delete Artifact

Delete registered artifacts from the registry. This will normally be utilized during testing but could be utilized during operations if a registration was made my mistake. Privilege for this capability should be limited. This use case pertains to the Data Engineer actor who will delete registered artifacts.

1. Data Engineer authenticates for access to the Registry service interface (include Security service Authenticate User use case [8]).
2. Data Engineer marks a registered artifact as deleted via the Registry service interface.
3. Registry service deletes the metadata associated with the artifact in the underlying metadata store.

Alternative: Operation Not Allowed
At step 3, the Registry service does not allow the operation per policy.

a. Registry service checks policy for allowed operations.
b. Registry service does not allow deletion of the artifact per policy.

3.8 Query Artifact

Discover registered artifacts from the registry by submitting queries against the registered metadata attributes. This use case pertains to the Data Engineer and Search service actors.

1. Search service submits a query for artifact(s) via the Registry service API.

2. Registry service accepts the query and returns metadata for one or more artifacts from the underlying metadata store matching the criteria.

4.0 REQUIREMENTS

The architecture definition phase of the PDS 2010 project resulted in the decomposition of the system into several elements [3]. The Registry service derives from the Catalog/Data Management element, which was derived from requirements 2.2.2 and 2.6 of the PDS Level 1, 2, and 3 Requirements document [1]. The following level 3 requirements are relevant to this service:

> **2.2.2** PDS will track the status of data deliveries from data providers through the PDS to the deep archive
> **2.6.2** PDS will design and implement a catalog system for managing information about the holdings of the PDS
> **2.6.3** PDS will integrate the catalog with the system for tracking data throughout the PDS
> **2.8.2** PDS will maintain a distributed catalog system which describes the holdings of the archive
> **2.8.3** PDS will provide standard protocols for locating, moving, and utilizing data, metadata and computing resources across the distributed archive, among PDS nodes, to and from missions, and to and from the deep archive

In addition to the level 4 and 5 requirements specified below, the Registry service must also comply with the general service-based requirements found in the General System SRD document [5].

4.1 Level 4 Requirements

The level four requirements in PDS represent subsystem or component requirements at a high level. The following requirements pertain to the Registry service:

L4.REG.1 - The system shall maintain distributed registries of artifacts. (2.6.2, 2.8.2)

> Ideally, each PDS Node that maintains a repository of data will have a corresponding registry.

L4.REG.2 - The system shall federate the registries. (2.8.2)

> To federate is to form a single centralized unit from a number of entities, within which each keeps some internal autonomy.

L4.REG.3 - The system shall register artifacts of a data delivery into an instance of the registry. (2.2.2, 2.6.2)

> A data delivery consists of artifacts including but not limited to data, document and software.

L4.REG.4 - The system shall allow for management of the metadata associated with registered artifacts. (2.6.2)

4.2 Level 5 Requirements

The level five requirements in PDS represent subsystem or component requirements at a detailed level. The following requirements pertain to the Registry service:

L5.REG.1 - The service shall accept artifact registrations. (L4.REG.3, UC 3.2)

L5.REG.2 - The service shall provide a means for relating artifact registrations. (L4.REG.3, UC 3.2)

> This allows for the equivalent of batch registrations and enables further operations (e.g., approve, delete, etc.) on all artifacts within a batch.

L5.REG.3 - The service shall maintain policy regarding the classes of artifacts to be registered. (L4.REG.1, UC 3.1)

> The service will capture and store a common set of metadata elements for each registered artifact. The policy will also include specification of metadata elements beyond the common set for each class of artifact where necessary.

L5.REG.4 - The service shall accept metadata for a registered artifact in a defined format. (L4.REG.3, UC 3.2)

> The defined format of the metadata is likely an XML structure governed by an associated XML Schema.

L5.REG.5 - The service shall validate metadata for a registered artifact. (L4.REG.3, UC 3.2)

L5.REG.6 - The service shall assign a global unique identifier to a registered artifact. (L4.REG.3, UC 3.2)

L5.REG.7 - The service shall assign a version to a registered artifact based on its logical identifier. (L4.REG.3, UC 3.2)

L5.REG.8 - The service shall store metadata for a registered artifact in an underlying metadata store. (L4.REG.3, UC 3.2)

L5.REG.9 - The service shall allow updates to registered artifacts. (L4.REG.4, UC 3.3)

L5.REG.10 - The service shall allow approval of registered artifacts. (L4.REG.4, UC 3.4)

Initial registrations result in an artifact being in an unapproved state. The meaning of artifact approval requires definition for PDS.

L5.REG.11 - The service shall allow deprecation of registered artifacts. (L4.REG.4, UC 3.5)

Similar to the approved state, the meaning of artifact deprecation still requires definition for PDS.

L5.REG.12 - The service shall allow undeprecation of registered artifacts. (L4.REG.4, UC 3.6)

L5.REG.13 - The service shall allow deletion of registered artifacts. (L4.REG.4, UC 3.7)

L5.REG.14 - The service shall allow queries for registered artifacts. (L4.QRY.1, UC 3.8)

The L4.QRY.1 requirement resides in the Search Service SRD/SDD [8].

L5.REG.15 - The service shall enable replication of registry contents with another instance of the service. (L4.REG.2)

L5.REG.16 - The service shall enable verification of registry contents. (L4.REG.2)

Verification includes checking for registered artifact existence and verifying the checksum.

5.0 DESIGN PHILOSOPHY, ASSUMPTIONS, AND CONSTRAINTS

The intent of the Registry service is to provide a generic and simple solution for registering artifacts within the system. Although the service facilitates capabilities for tracking and search, the Registry service does not ultimately satisfy those requirements. Those requirements are satisfied by the Monitor and Search services, respectively.

The design of this service heavily leverages current work efforts by CCSDS in the form of the Registry and Repository Reference Model [6]. This reference model in turn, heavily leverages the ebXML suite of standards managed by OASIS.

6.0 ARCHITECTURAL DESIGN

The architectural design covers the component breakdown within the service, external/internal interfaces and the associated data model.

6.1 Component Architecture

The following diagram details the architecture for the Registry service:

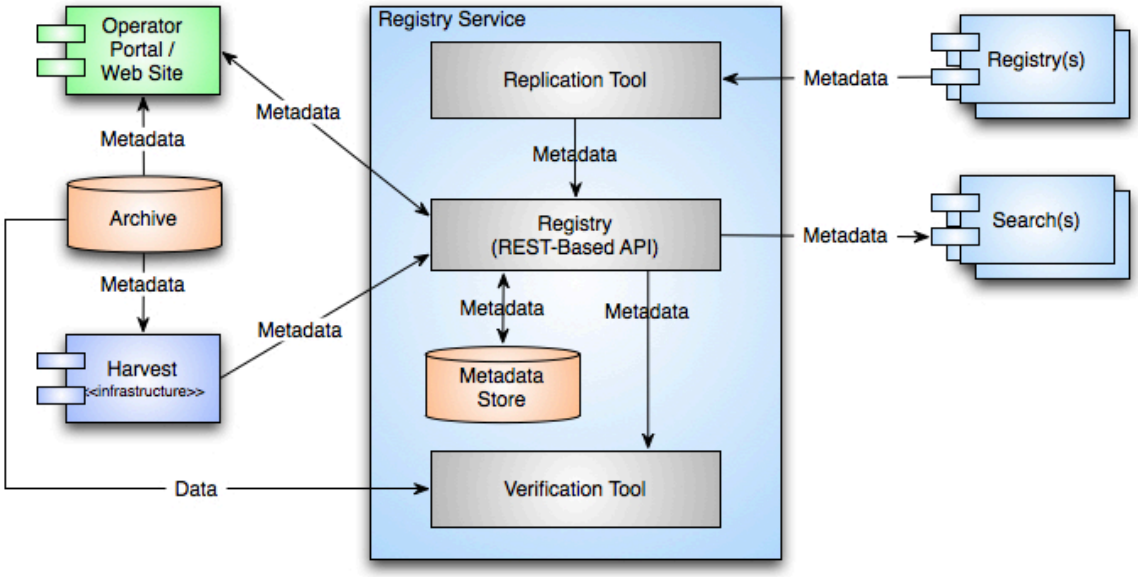

Figure 3: Registry Service Architecture

The service architecture provides for two scenarios for populating a registry:

Ad hoc Access via Portal
Although this is somewhat of a misnomer because the portal will use the REST-based API to access the service, this is where the Data Engineers can perform ad hoc registrations as well as the perform functions like approve and deprecate which are probably not suitable for automated access. Ad hoc access also includes performing functions like query for the purposes of managing the registry.

Automated Access via API
This scenario represents access from services like Harvest and Ingest, where registrations are automated and achieved through service-to-service communication via the REST-based API.

The diagram above assumes that the registered artifact resides in a managed repository (i.e., archive directory structure) and will be registered in place. The

following diagram supports the scenario where the Storage service is utilized to manage the physical bits of the registered artifact:

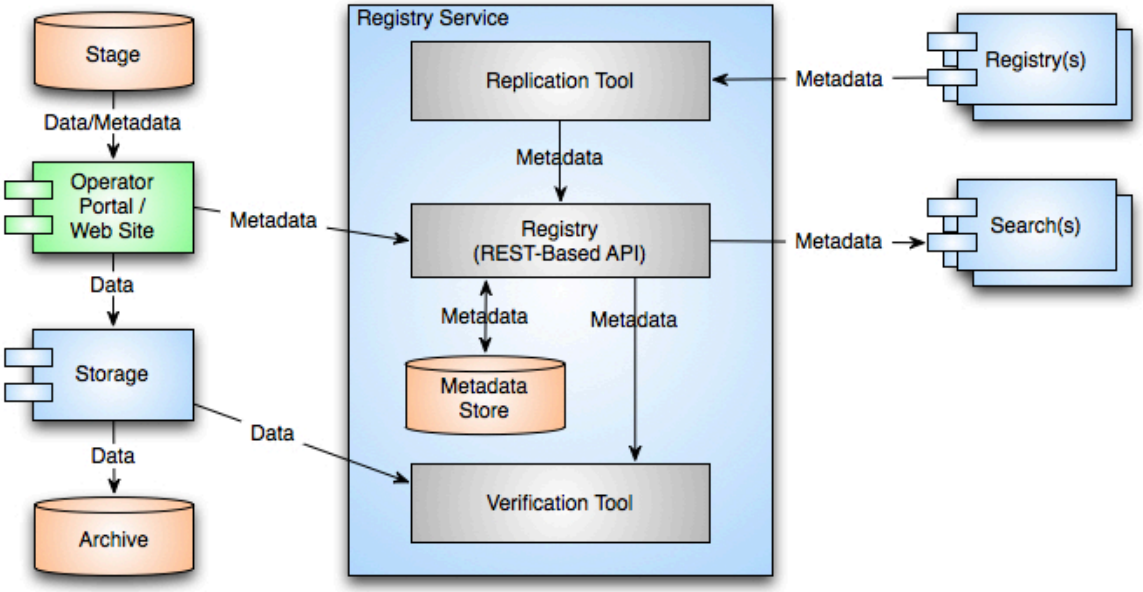

Figure 4: Registry Service Architecture (with Storage)

This scenario mainly pertains to the management of schemas and other documents within the system that will not reside in a Node's archive directory structure. In this case the Operator Portal submits the files to the Storage service and then registers those files as an artifact with the Registry service.

In addition to population of the registry, there are two scenarios for importing/exporting metadata from the registry:

Metadata Import for Replication
There are two purposes for replication. The first is to populate an aggregate registry utilized for satisfying tracking, metrics reporting and catalog-level search requirements. The second is for sharing artifact registrations between Nodes. The Replication Tool pulls artifact registrations from other Registry service instances according to its local configuration.

Metadata Export for Search
This is where the Registry service facilitates end-user search. Instances of the Search service will query one or more instances of the Registry service in order to generate search indices. These indices are tailor-able for the search application that will utilize them.

In addition to registry population and metadata export, the service will also provide the capability to perform verification for registered artifacts. This capability is intended to be executed local to the registry or more specifically,

local to the repository associated with the registry. A capability like this could utilize a lot of bandwidth if executed remotely.

The following diagram details the “big picture” architecture of the Registry service and depicts a possible deployment scenario for service instances:

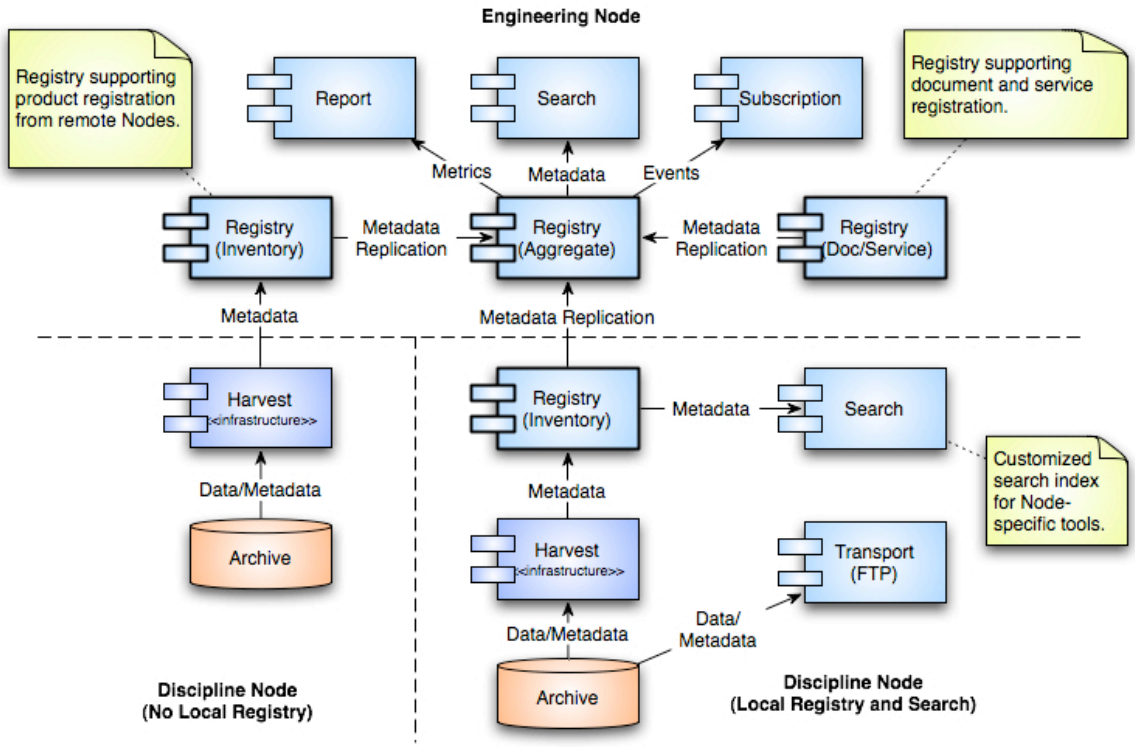

Figure 5: Registry Service Architecture (Big Picture)

The diagram above depicts four instances of the Registry service within the system and lends some insight to the deployment of the service. The instances are as follows:

Local Node Registry
The plan is to have a local instance of the Registry service installed at each Node that hosts a local repository. A local instance of the Harvest service configured for the local repository populates this registry. A local Search service extracts metadata from this registry to support Node-specific search tools.

Remote Node Registry
Although this is not the preferred deployment, a centralized instance of the Registry service is available that Nodes can populate remotely utilizing a local instance of the Harvest service.

Centralized Registry
The plan is to have a centralized registry for managing schema and service registrations. The Operators and Data Engineers use the Operator Portal to populate this registry.

Aggregate Registry
The aggregate registry instance will contain replicated registry entries from all other Registry service instances. This registry will allow the system to satisfy requirements for catalog-level search, metrics generation and subscription notification without the need to perform live queries across the distributed registry instances. Replications to the aggregate registry and index generation are performed during off-peak hours further increasing productivity of the system.

6.2 Interface Design

The following diagram focuses on the interfaces, both external and internal for the Registry service:

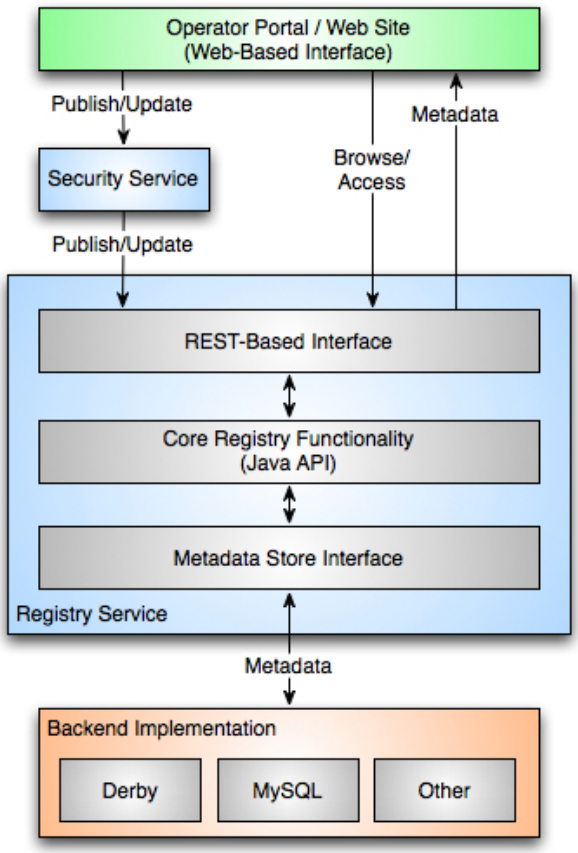

Figure 6: Registry Service Interfaces

The interfaces are described in more detail in the following sections.

6.2.1 External Interface Design

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

- A URL assigned to every resource
- Formulate URLs in a predictable manner
- Use HTTP methods for actions on a resource (GET, POST and DELETE)
    - Due to similarities between POST and PUT, the design team decided to utilize POST exclusively.
- Leverage HTTP protocol headers and response codes where applicable

The goals for the interface are as follows:

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

In addition, each interface should adhere to the following:

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

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

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

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

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

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

6.2.2 Internal Interface Design

The primary internal interface for the Registry service involves communication with the underlying metadata store. This interface will follow a generic design with the intent of supporting multiple backend implementations for the metadata store. The layered design for the backend implementation allows for technology refresh and multiple deployment scenarios. The metadata store interface will support the data model detailed in the following section of this document.

6.3 Data Model

The following diagram represents the CCSDS Registry logical model (key classes) and is the basis for implementing the underlying metadata store for this service:

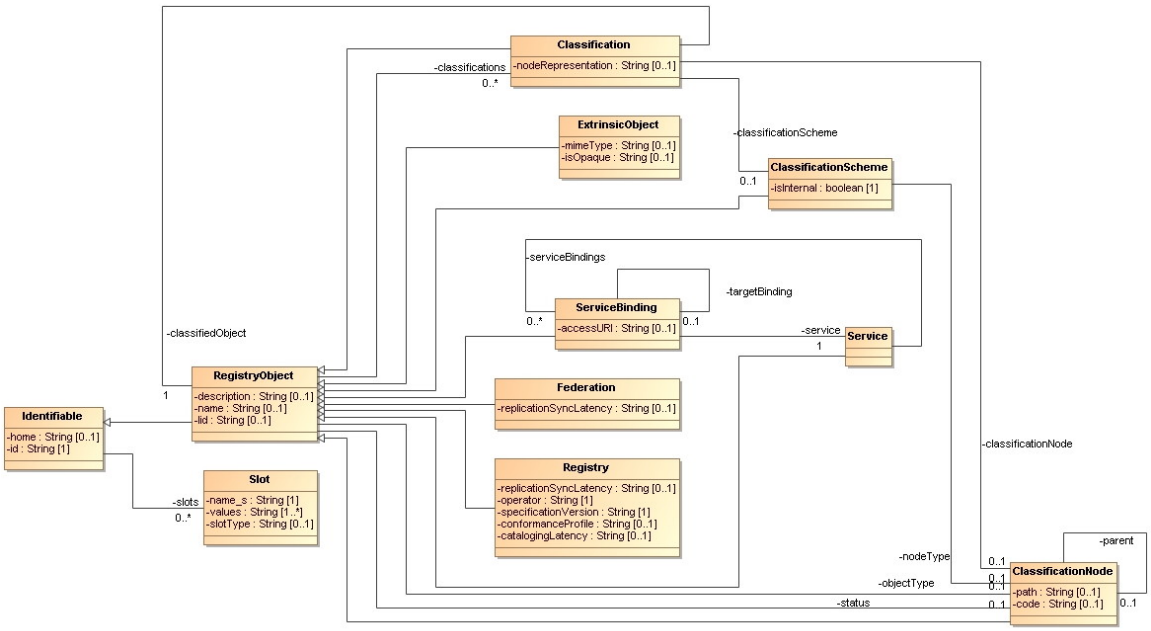

Figure 7: Registry Service Data Model

The classes detailed in the diagram above and a couple of others that are important to the design of the Registry service are defined below:

Association (not pictured)
Specifies a relationship between two RegistryObject instances. A PDS example of an association is that a data product is a member of a collection.

AuditableEvent (not pictured)
Instances of AuditableEvent record the actions taken against a RegistryObject instance. For example, approval or deprecation of a RegistryObject is an auditable event.

Classification
Specifies the classification of a RegistryObject utilizing the ClassificationScheme and ClassificationNode classes. Classifications utilized by the Registry service are defined in the PDS4 data model [7].

ExtrinsicObject
This is the place holder object for PDS products in the data model. All PDS products (e.g., data products, investigations, instruments, personnel, etc.) will be derived from the ExtrinsicObject class. The PDS products are defined in the PDS4 data model [7].

Federation
An instance of the Registry service may belong to a federation of registries. There is likely to be one federation defined for the PDS Registry service instances.

Identifiable
This class provides the ability to identify objects by an id attribute and is the parent class for all of the classes defined here.

Registry
Represents an instance of a Registry service within the PDS.

RegistryObject
The RegistryObject class extends the Identifiable class and serves as a common super class for most classes in the data model. The term “artifact”, used throughout this document, is equivalent to an instance of the RegistryObject class.

Service
This class captures descriptions of services utilizing the ServiceBinding class.

Slot

The Slot class provides a dynamic way to add arbitrary attributes to RegistryObject instances. For example, this is where the PDS will capture the 10 plus or minus 2 keywords to be utilized in global search scenarios.

7.0 ANALYSIS

The early efforts for the Registry service looked into the use of registries in existing science data systems (PDS, SPASE, IVOA, ECHO, etc.) and other entities (OGC, OASIS). The Registry Services document [9], initially authored by Mike Martin and contributed to by the Distributed Infrastructure Design Team, captures the details of that survey. Two prevailing registry standards were identified in that survey:

UDDI (Universal Description Discovery & Integration)
UDDI is one of the standards from the WS-*(Web Services) stack of standards (e.g., SOAP, WSDL, etc.). It promotes a service registry or "yellow pages" of available services.

ebXML (Electronic Business using eXtensible Markup Language)
The ebXML standard is a modular suite of specifications enabling business of the Internet. It promotes a registry as an information repository and supports registration of different objects based on a Registry Information Model (ebRIM) profile per object type.

Although they both facilitate a SOA, the ebXML standard better facilitates the federated registry concept. The following benefits of ebXML include:

- Provides a standard way to manage information assets
- Manages user-defined organization of and relationships among content and metadata
- Enforces user-defined standards for content
- Includes capabilities for managing and governance of information asset lifecycles
- Provides flexible mechanisms for content delivery
- Manages secure access to information assets
- Facilitates event-based delivery of information to appropriate personnel or systems
- Enables integration of information assets across organizational boundaries

With that conclusion, the development team evaluated two ebXML-based software packages:

freebXML
The freebXML package is open source and available as a free download. The team successfully installed the package after a few failed attempts. The package did support product registration but would require additional development to meet the rest of the PDS requirements. In addition, support for the package was not active and would require the PDS to

essentially continue to develop and maintain the package. Another drawback was that the package conformed to version 2 of the standard. The current version is an older version.

WellGEO RegRep from Wellfleet Software Corporation
This is a Commercial Off-The-Shelf (COTS) package developed and distributed by the main author of freebXML. The team worked with the author to setup a prototype installation of the software that did perform to expectations. The first caveat with the prototype was that it required quite a bit of custom coding and apparent patches to the package to meet our requirements. The impression from this was that the software was not very mature. The second caveat was that the estimated cost of nearly a million dollars for the first year with maintenance in the following years exceeded PDS budget constraints.

After these two evaluations, the team decided to take a close look at the CCSDS reference model [6] and implement a conformant Registry service that supports the PDS requirements. Although the reference model is a work in progress, PDS is contributing to the effort by developing a reference implementation.

8.0 IMPLEMENTATION

The PDS 2010 system is a phased implementation with increasing capabilities delivered in three planned builds. The builds are as follows:

- **Build 1** – This build consists of the Ingestion subsystem including the Security, Harvest, Registry (Inventory, Dictionary, Document, Service) and Report components along with the Data Provider tool suite.
- **Build 2** – This build consists of the Distribution subsystem including the Search and Monitor components along with a revised web site and general portal applications.
- **Build 3** – This build consists of enhanced user capabilities include the Order and Subscription components along with integration of Discipline Node applications and science services.

The Registry service is scheduled for delivery in Build 1. This initial delivery will support test collection generation and registration. Additional capabilities are planned for follow-on deliveries as testing progresses and the data model matures.

The implementation platform for the Registry service is the Java 2 Platform Standard Edition 6.0. Implementation of the REST-based interface will utilize Jersey, which is a reference implementation of the Java API for RESTful Web Services (JAX-RS) framework. In addition, development will utilize publicly available libraries for interface development, message handling and file system access.

Figure 4 above details the scenarios for deployment of the Registry service instances. The preferred scenario for Node deployment is to run an instance of the Registry service and an instance of the Harvest tool on a single machine locally at the Node. Service packaging consists of a Web Application Archive (WAR), which requires an Application Server (e.g., Apache Tomcat) installed on the target machine to host the service. The following diagram depicts this deployment scenario:

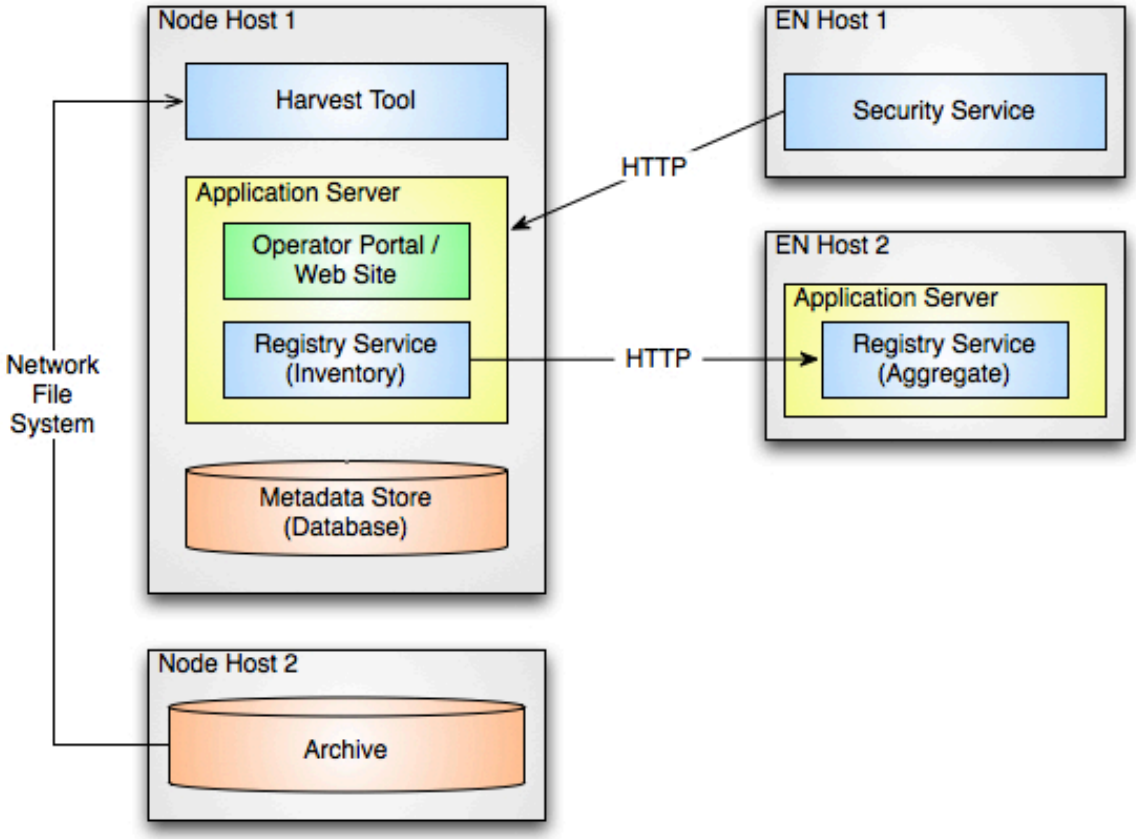

Figure 8: Registry Service Deployment

The Harvest tool accesses the Node's PDS archive data repository via a network file system. The phrase "network file system" is used generically here since the actual implementation may vary from Node to Node based on the choice of platform available at the Node. A local database server instance (e.g., MySQL) is required to serve as the metadata store for the registry. Communication between the Application Server and the Security service for authentication and authorization is accomplished using the using the Hypertext Transfer Protocol (HTTP). Replicated product registrations are pulled from the Node's Registry service instance to the aggregate Registry service instance at the EN via the REST-based interface using HTTP.

9.0 DETAILED DESIGN

This section offers a more detailed look at certain aspects of the Registry service design. The following diagram details how the status of a registered artifact changes state:

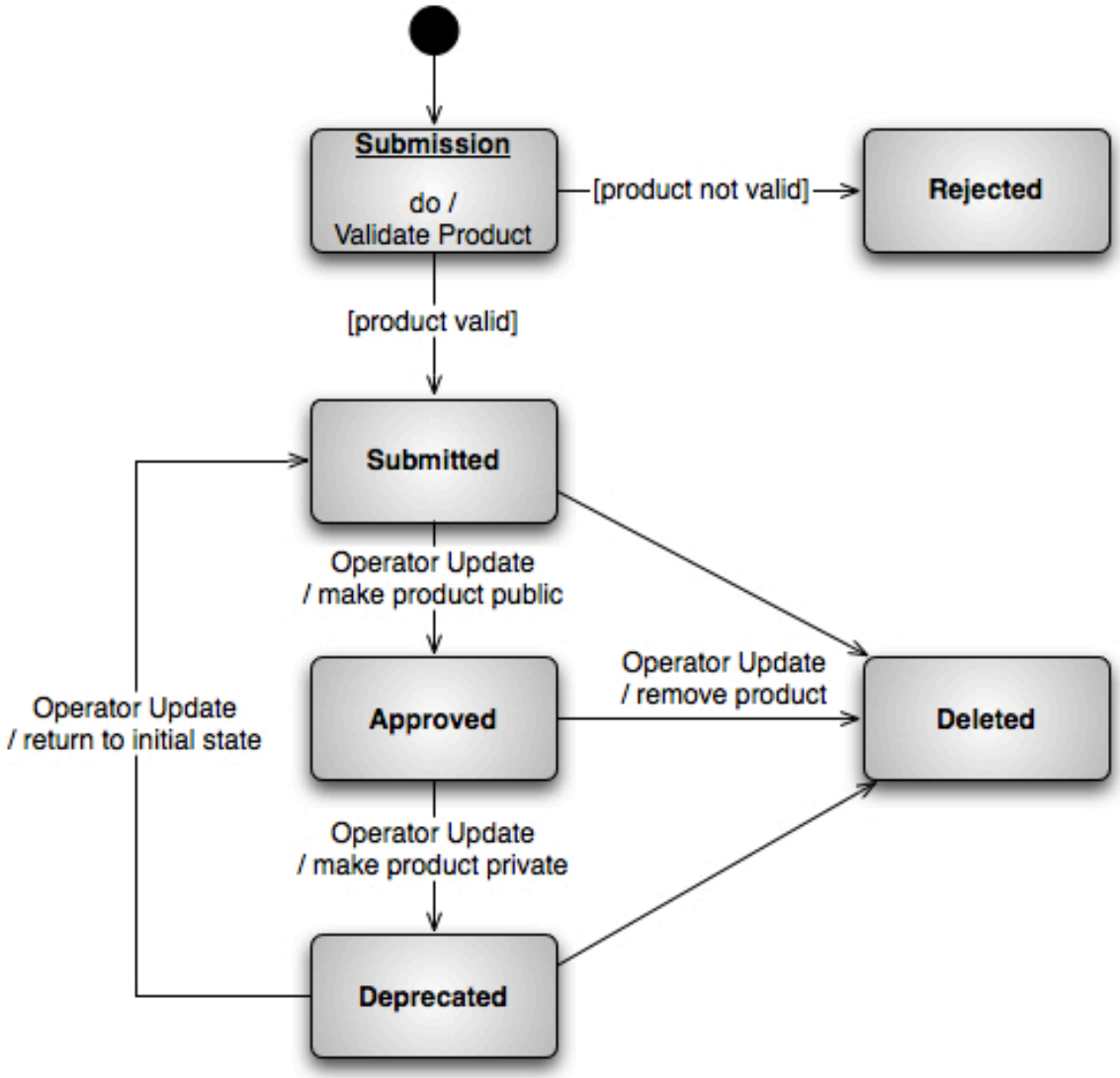

Figure 8: Registry Service State (Product Status)

The status of “Submitted” is considered the initial state for a successfully registered artifact. The Operator initiates all other changes in artifact status. Changing an artifact’s status to “Deprecated” is only allowed if the current status is “Approved” and undeprecating an artifact returns it to the “Submitted” status.

APPENDIX A ACRONYMS

The following acronyms pertain to this document:

API	Application Programming Interface
CCSDS	Consultative Committee for Space Data Systems
ebRIM	ebXML Registry Information Model
ebXML	Electronic Business using XML
ECHO	EOS ClearingHOuse
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IVOA	International Virtual Observatory Alliance
JAX-RS	The Java API for RESTful Web Services
JPL	Jet Propulsion Laboratory
JSON	JavaScript Object Notation
NASA	National Aeronautics and Space Administration
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
PDS	Planetary Data System
REST	Representational State Transfer
SDD	Software Design Document
SOA	Service Oriented Architecture
SPASE	Space Physics Archive Search and Extract
SRD	Software Requirements Document
UC	Use Case
UDDI	Universal Description Discovery & Integration
WADL	Web Application Description Language
WAR	Web Application Archive
WSDL	Web Service Definition Language
XML	Extensible Markup Language