Planetary Data System

Registry Service

Software Requirements and Design Document (SRD/SDD)

Sean Hardman
Paul Ramirez

March 6, 2010
Version 0.3

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

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 products consisting of data files and label files, schemas, dictionary definitions for objects and elements, 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, August 2006.

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

[3] Planetary Data System (PDS) 2010 System Architecture Specification, Version 1.0, February 28, 2010.

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

[5] Planetary Data System (PDS) Service Software Requirements Document (SRD), TBD.

1.5 Applicable Documents

[6] CCSDS Registry and Repository Reference Model, June 25, 2009.

[7] PDS4 Information Model Specification, PDS4 Information Model Specification Team, February 2010.

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 SERVICE 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). In addition to registration, this service will provide the means to store artifacts when a given operational scenario calls for it. The intent of this service is to facilitate tracking, auditing and maintenance of artifacts within PDS (e.g., products, dictionary definitions, schemas, services, etc.). The following diagram details the context of the Registry service, represented as the Inventory, Document and Service services, within the system:

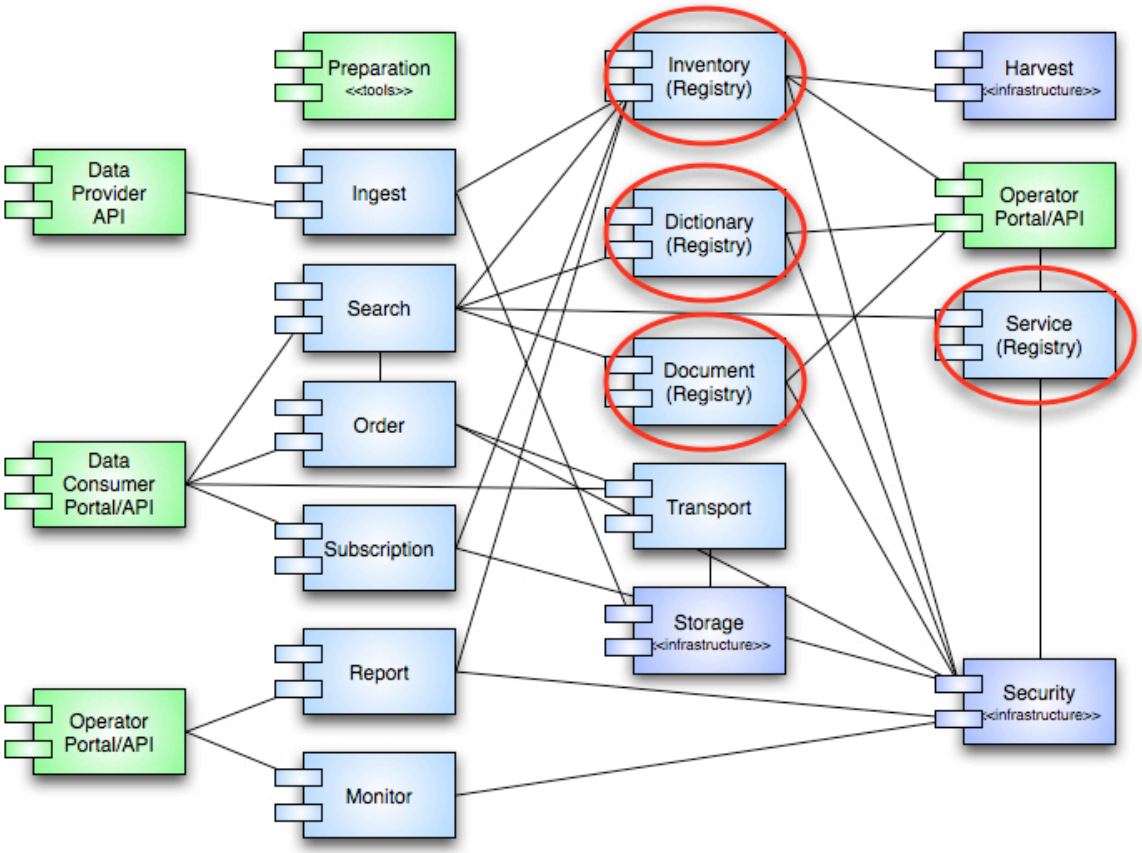

Figure 1: Registry Service Context

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 about something of interest within the system. This type of entry in the existing PDS infrastructure is akin to descriptions captured for missions, instruments, data sets, targets, 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.). Depending on the class of registered artifact, support for storage (i.e., repository) may be internal, external, or both.

Relationship Entry
This type of entry will serve as a means to tie registered artifacts together. Such support is necessary for example to correlate data set descriptions to a set of products. These relationships may span registries and thus the need for coordination amongst registries exists.

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 product registry would not seek to replace the nodes existing catalog but act as the infrastructure component equivalent. The following is an accounting of registries that would be available within the system:

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

The Data Engineers at the Engineering Node perform ingestion of catalog metadata into the catalog database. The PDS 2010 Inventory service will provide similar functionality for the catalog and product level metadata. This instance is a specialization of a registry that merely captures catalog-level descriptions and relationships.

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 instance of the service.

Document
With the transition to XML, management of schemas, which govern XML instance files (e.g. labels), becomes of utmost importance. Schemas must be captured and readily available and this registry will provide this role. Given that schema typically exist in the form of a file, this particular registry requires a repository for storing the files.

Service
The service registry will capture 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 may include a repository if the operational scenario and/or requirements call for it.

3.0 USE CASES

A use case represents a capability of the service 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 service but interacts with the service. 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:

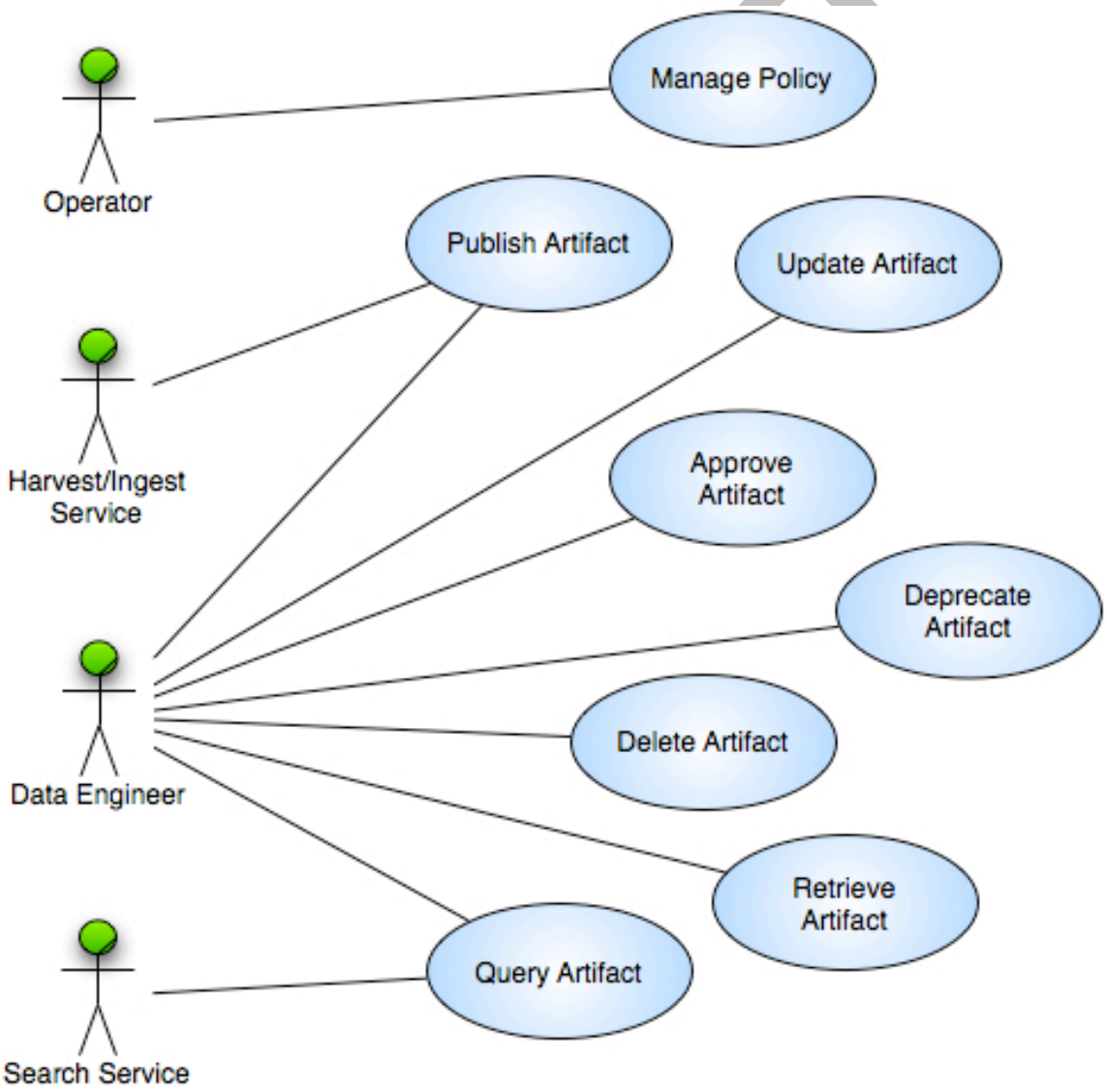

Figure 2: Registry Service Use Cases

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 products.

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 artifacts.

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).
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 data store) the operation.

3.2 Publish Artifact

Register artifacts within 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).
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 data store.
6. Registry service stores the artifact in the underlying repository.

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).
b. Data Engineer submits an artifact for registration via the Registry service interface.
c. Return to primary scenario at step 3.

Alternative: Reference Registration
At step 6, the artifact is simply referenced and not stored in the repository.

a. Registry service stores a reference to the artifact in its home repository.

3.3 Update Artifact

Replace registered artifacts or their associated metadata. This use case pertains to the Data Engineer who will perform replacement registrations of artifacts within the system.

1. Data Engineer authenticates for access to the Registry service interface (include Security service Authenticate User use case).
2. Data Engineer submits a replacement 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 data store.
5. Registry service stores the artifact in the underlying repository.
6. Registry service deletes the original artifact from the underlying repository.

Alternative: Reference Registration
At step 5, the artifact is simply referenced and not stored in the repository.

a. Registry service stores a reference to the artifact in its home repository.
b. Registry service deletes the original reference to the artifact from the underlying repository.

Alternative: Metadata Update
At step 2, only the metadata is updated.

a. Data Engineer submits replacement metadata for a registered artifact.
b. Registry service validates the submitted metadata.
c. Registry service records the metadata associated with the artifact in the underlying data 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).
2. Data Engineer marks a registered artifact as approved via the Registry service interface.
3. Registry service records the approval in the underlying data 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).
2. Data Engineer marks a registered artifact as deprecated via the Registry service interface.
3. Registry service records the deprecation in the underlying data store.

3.6 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).
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 data store.
4. Registry service deletes the artifact from the underlying repository.

<u>Alternative: Operation Not Allowed</u>
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.

<u>Alternative: Reference Registration</u>

At step 4, the artifact is simply referenced and not stored in the repository.

a. Registry service deletes the reference to the artifact from the underlying repository.

3.7 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 data store matching the criteria.

3.8 Retrieve Artifact

Retrieve registered artifacts from the registry. The Registry service is not intended to be the public interface for retrieval of artifacts from PDS. This use case pertains to the Data Engineer.

1. Data Engineer submits a request to retrieve a registered artifact identified by its global identifier.
2. Registry service accepts the request and returns the registered artifact from the underlying repository.

Alternative: Reference Registration

At step 2, the artifact is simply referenced and not stored in the repository.

a. Registry service accepts the request and returns the registered artifact from its home repository.

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 accessing data, metadata and computing resources across the distributed archive

In addition to the Inventory service, the Dictionary, Document and Service (Registry) services are instances of the Registry service.

4.1 Level 4 Requirements

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

L4.REG.1 - The system shall maintain distributed registries of artifacts. (2.2.2, 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 products (artifacts) including but not limited to data, document and software.

L4.REG.4 - The system shall provide application programming interfaces for managing and accessing the content of the registries. (2.6.3, 2.8.3)

4.2 Level 5 Requirements

The level five requirements in PDS represent subsystem, component or tool 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.2, 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 unique identifier. (L4.REG.3, UC 3.2)

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

L5.REG.9 - The service shall store the registered artifact in an underlying repository where appropriate. (L4.REG.3, UC 3.2)

> Some artifacts will be registered as a reference to the artifact in its home repository.

L5.REG.10 - The service shall allow replacement of registered artifacts. (UC 3.3)

L5.REG.11 - The service shall allow approval of registered artifacts. (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.12 - The service shall allow deprecation of registered artifacts. (UC 3.5)

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

L5.REG.13 - The service shall allow deletion of registered artifacts. (UC 3.6)

For artifacts that were registered as references to an artifact in its home repository, deletion will not delete the actual artifact from its home repository.

L5.REG.14 - The service shall allow queries for registered artifacts. (UC 3.7)

L5.REG.15 - The service shall allow retrieval of registered artifacts. (UC 3.8)

L5.REG.16 - The service shall provide an application programming interface. (L4.REG.4, UC 3.1 – 3.8)

L5.REG.17 - The service shall restrict access to operations that alter the content of the registry. (UC 3.1, UC 3.2, UC 3.3, UC 3.4, UC 3.5, UC 3.6)

L5.REG.18 - The service shall enable replication of registry contents with another instance of the service. (TBD)

L5.REG.19 - The service shall enable verification of registry contents. (TBD)

Verification includes checking for registry 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 work current efforts by CCSDS in the form of the Registry and Repository Reference Model [5]. Their 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 Service Architecture

The following diagram details the architecture for the Registry service:

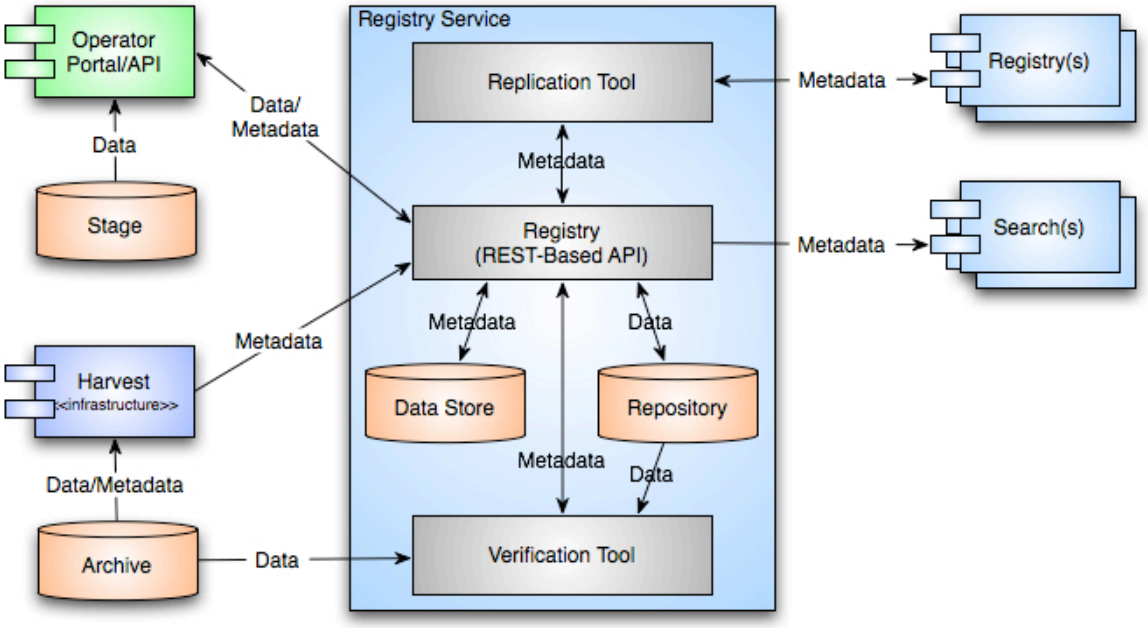

Figure 3: Registry Service Architecture

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 and retrieval 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.

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

Replication
There are two purposes for replication. The first is to populate an aggregate registry utilized for satisfying tracking, metrics and catalog-level search requirements. The second is for sharing registered artifacts between Nodes.

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:

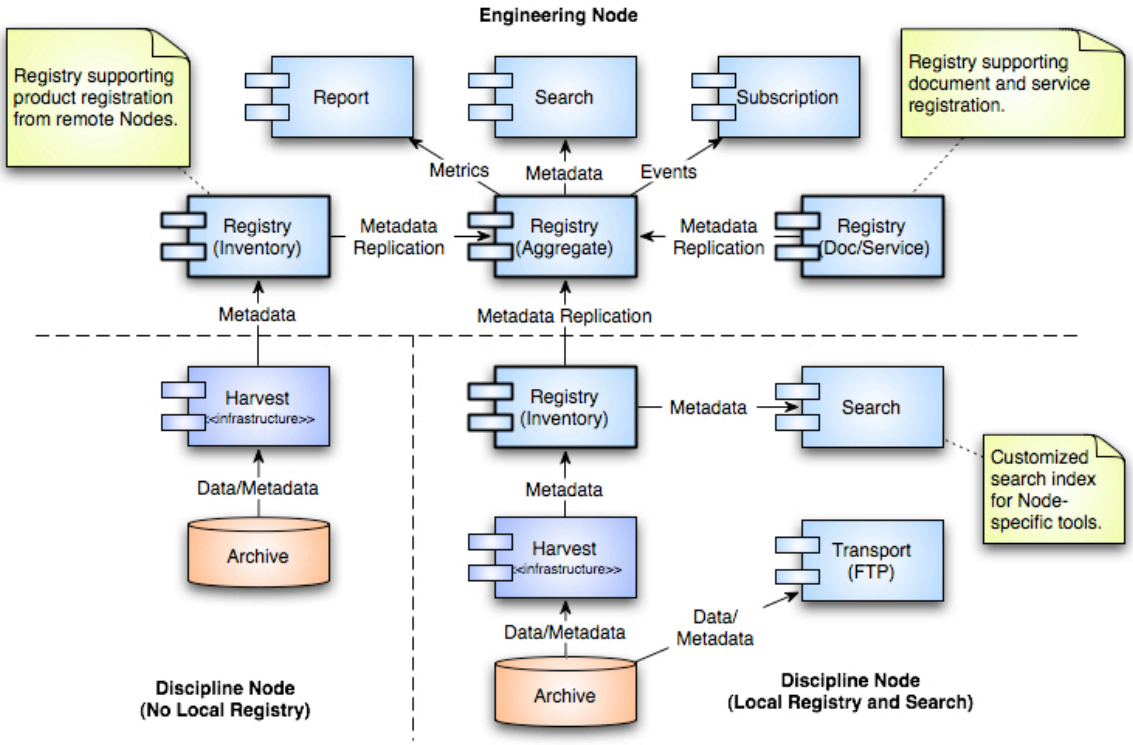

Figure 4: Registry Service Architecture (Big Picture)

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. Replication to the aggregate registry and index generation are performed during off-peak hours further increasing productivity of the system.

6.2 External Interface Design

The Registry service offers a REST-based external interface. 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
- 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 JSON
- Allow for extensible as new types of artifacts are defined

Any interface that modifies the contents of the registry will incorporate security. This means that any interface specified below as an HTTP POST or PUT will first require interaction with the Security service. The only change to these interfaces will be in terms of a required HTTP header or cookie being set that will provide the underlying registry with a means to verify the validity of the request. More information on this topic will be forthcoming in a subsequent version of this document.

See Appendix B for interface details.

6.3 Internal Interface Design

The internal interfaces for the Registry service involve communication with the underlying data store and repository.

More information on this topic will be forthcoming in a subsequent version of this document.

6.4 Data Model

The following diagram represents the CCSDS Registry model and is the basis for implementing the underlying data store for this service:

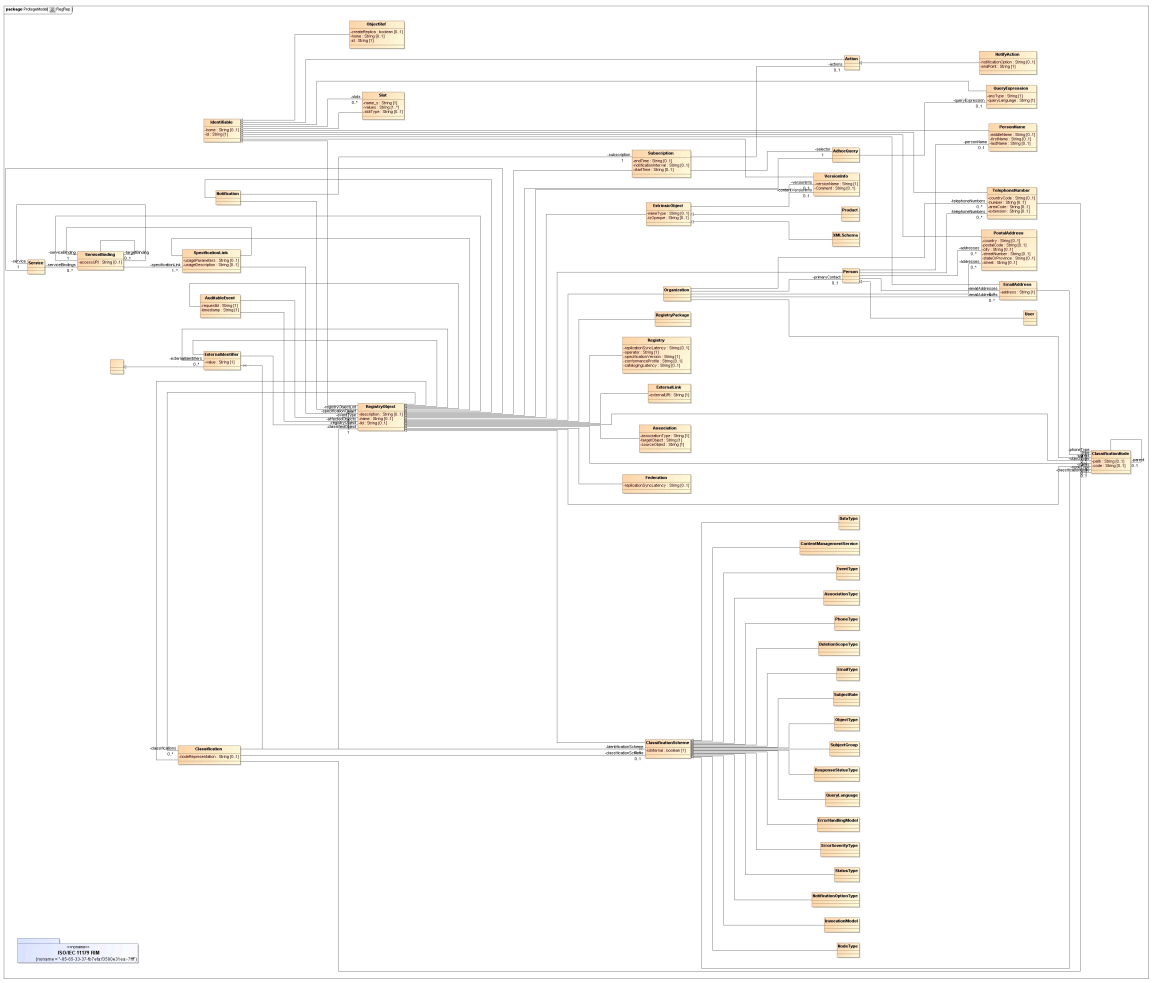

Figure 5: Registry Service Data Model

The data model includes specific classes for the registry, federation and service. All other registered items are extrinsic objects.

More information on this topic will be forthcoming in a subsequent version of this document.

7.0 ANALYSIS

The early efforts for the Registry service looked into registry-based standards. The two prevailing registry standards are:

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. Based on that decision, the development team evaluated two available 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 an older version of the standard.

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 specification [6] and implement a conformant Registry service that supports the PDS requirements.

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. There is no planned phasing with regard to the implementation with all planned capabilities available in Build 1.

The implementation platform for the Registry service is the Java 2 Platform Standard Edition 6.0. In addition, development will utilize publically available libraries for interface development, message handling and file system access.

Figure 4 above details the scenarios for deployment. 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.

9.0 DETAILED DESIGN

More information on this topic will be forthcoming in a subsequent version of this document.

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
HTTP	Hypertext Transfer Protocol
JAX-RS	The Java API for RESTful Web Services
JPL	Jet Propulsion Laboratory
NASA	National Aeronautics and Space Administration
OASIS	Organization for the Advancement of Structured Information Standards
PDS	Planetary Data System
REST	Representational State Transfer
SDD	Software Design Document
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

APPENDIX B REST-BASED INTERFACE

The generated documentation that follows came directly from the Registry service source code using the Java API for RESTful Web Services (JAX-RS) framework.

PDS Registry Service API

This is the proposed REST interface for a PDS Registry.

Resources

/registry

This is the root resource for managing the registry.

Methods

/registry/synch

Methods

PUT

Synchronizes the incoming registry artifacts with those already present in the registry.

acceptable request representations:

- application/xml
- application/json

/registry/status

Methods

GET

Retrieve the status of the registry service. This can be used to monitor the health of the registry.

available response representations:

- Status Code 200 - application/xml (ns3:status_information)

/registry/artifacts

This interface delegates all functions involving an artifact. This is defined as a sub-resource to the registry resource merely to partition off the operations involving artifacts.

Methods

GET

Allows access to all the artifacts managed by this repository. This will need to be converted over to a paged list

available response representations:

- application/xml
- application/json

POST

Publishes an artifact to the registry. Publishing includes validation, assigning a version, storage (if object contents are attached), and notification. Validation from the perspective of publishing is on the metadata provided with the entry.

acceptable request representations:

- application/xml
- application/json

available response representations:

- */*

/registry/artifacts/{lid}/{version}

resource-wide template parameters

parameter	value	description
lid	*string*	
version	*string*	

Methods

GET

Retrieves an artifact from the registry. The local identifier with the version uniquely identifies one artifact.

available response representations:

- application/xml
- application/json

PUT

Updates the artifact in the registry with the lid and version to the given artifact

acceptable request representations:

- application/xml
- application/json

available response representations:

- application/xml
- application/json

DELETE

/registry/artifacts/{lid}/{version}/approve

resource-wide template parameters

parameter	value	description
lid	*string*	local identifier which identifies a unique set of artifacts
version	*string*	of the artifact's local identifier

Methods

PUT

This will set the status of a registered artifact to approved.

available response representations:

- application/xml
- application/json

/registry/artifacts/{lid}/{version}/deprecate

resource-wide template parameters

parameter	value	description
lid	*string*	local identifier which identifies a unique set of artifacts
version	*string*	of the artifact's local identifier

Methods

PUT

This will set the status of a registered artifact to deprecated.

available response representations:

- application/xml
- application/json

/registry/associations

Methods

GET

Retrieves all associations managed by the registry. This needs to be switched over to a paged response as it is likely to grow to a large set of associations.

available response representations:

- application/xml
- application/json

POST

Creates a new association between artifacts in the registry. The association has a source, destination, and is named.

acceptable request representations:

- application/xml
- application/json

available response representations:

- */*

/registry/associations/source/{lid}/{version}

resource-wide template parameters

parameter	value	description
lid	*string*	local identifier of the source artifact
version	*string*	of the given local identifier

Methods

GET

Retrieves all associations where the identified artifact is the source of the relationship.

available response representations:

- application/xml
- application/json

/registry/associations/source/{lid}/{version}/{relationship}

resource-wide template parameters

parameter	value	description
lid	*string*	local identifier of the source artifact
relationship	*string*	that exists between the source and target
version	*string*	of the given local identifier

Methods

GET

Retrieves all named associations where the identified artifact is the source of the relationship.

available response representations:

- application/xml
- application/json

/registry/associations/target/{lid}/{version}

resource-wide template parameters

parameter	value	description
lid	*string*	local identifier of the source artifact
version	*string*	of the given local identifier

Methods

GET

Retrieves all associations where the identified artifact is the target of the relationship.

available response representations:

- application/xml
- application/json

/registry/associations/target/{lid}/{version}/{relationship}

resource-wide template parameters

parameter	value	description
lid	*string*	local identifier of the source artifact
relationship	*string*	that exists between the source and target
version	*string*	of the given local identifier

Methods

GET

Retrieves all named associations where the identified artifact is the target of the relationship.

available response representations:

- application/xml
- application/json

/registry/associations/all/{lid}/{version}

resource-wide template parameters

parameter	value	description
lid	*string*	local identifier of the source artifact
version	*string*	of the given local identifier

Methods

GET

Retrieves all associations where the identified artifact is part of irregardless if it is the source or target.

available response representations:

- application/xml
- application/json

/registry/associations/all/{lid}/{version}/{relationship}

resource-wide template parameters

parameter	value	description
lid	*string*	local identifier of the source artifact
relationship	*string*	that exists between the source and target
version	*string*	of the given local identifier

Methods

GET

Retrieves all named associations where the identified artifact is part of irregardless if it is the source or target.

available response representations:

- application/xml
- application/json

/registry/storage

Methods

GET

Lists out all the containers managed by the store

available response representations:

- application/xml
- application/json

/registry/storage/{container}

resource-wide template parameters

parameter	value	description
container	*string*	

Methods

GET

available response representations:

- application/xml
- application/json

Representations

application/xml

application/json

Status Code 200 - application/xml (ns3:status_information)

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<status_information storedDataObjects="25" registeredArtifacts="10" serverStarted="2010-02-23T10:52:15.166-08:00" status="OK" xmlns="http://registry.pds.na
```

XML Schema

Source:

```
<xs:element name="status_information">
    <xs:complexType>
      <xs:sequence/>
      <xs:attribute name="status" type="registryStatus"/>
      <xs:attribute name="serverStarted" type="xs:dateTime"/>
      <xs:attribute name="registeredArtifacts" type="xs:int"/>
      <xs:attribute name="storedDataObjects" type="xs:int"/>
    </xs:complexType>
  </xs:element>
```

application/xml

application/json

application/xml

application/json

/

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json

/

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json

application/xml

application/json