

# A Reference Architecture for Space Information Management

Chris A. Mattmann\* and Daniel J. Crichton.†

*Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109*

J. Steven Hughes‡ and Paul M. Ramirez§

*Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109 and*

Daniel C. Berrios\*\*

*University of California, Santa Cruz, NASA AMES Research Center, Moffett Field, CA 94035*

**We describe a reference architecture for space information management systems that elegantly overcomes the rigid design of common information systems in many domains. The reference architecture consists of a set of flexible, reusable, independent models and software components that function in unison, but remain separately managed entities. The main guiding principle of the reference architecture is to separate the various models of information (e.g., data, metadata, etc.) from implemented system code, allowing each to evolve independently. System modularity, systems interoperability, and dynamic evolution of information system components are the primary benefits of the design of the architecture. The architecture requires the use of information models that are substantially more advanced than those used by the vast majority of information systems. These models are more expressive and can be more easily modularized, distributed and maintained than simpler models e.g., configuration files and data dictionaries. Our current work focuses on formalizing the architecture within a CCSDS Green Book and evaluating the architecture within the context of the C3I initiative.**

## I. Introduction

Today's software systems are growing in complexity, dynamicity, heterogeneity, and becoming increasingly more costly to operate. Space data systems are no exception to this emerging trend; they are highly distributed, complex software entities that must manage information from the point of generation, during distribution via one of many existing CCSDS protocols (e.g., CFDP, Proximity-1), at arrival on Earth, during delivery to processing centers, distribution to mission personnel and scientists, and ultimately for long-term archiving. Space data systems should be developed and operated using *models* of the information that they manage. There are many different models that need to be managed across an end-to-end space data system. These should include scientific and engineering data models (e.g., models of images taken on a spacecraft, of control center operations), and even models of other models (*meta-models*). To avoid rigidity, however, software used by a space data system should be flexible: it should be driven by the models that it operates on, and not *vice versa*.

Current space data systems are not flexible, and are extremely tied to the information on which they operate. However, they are not unique in this regard. Bio-medical computing systems, science data processing systems, space flight operation systems all exhibit the same brittle design – software and model tied together – a change in the model requires a change in the software, a change in the software leads to a change in the model.

---

\* Staff Software Engineer, Modeling and Data Management Systems Section, 4800 Oak Grove Drive, M/S 171-264.

† Principal Computer Scientist, PDS Engineering Node Office, 4800 Oak Grove Drive.

‡ Principal Computer Scientist, Modeling and Data Management Systems Section, 4800 Oak Grove Drive, M/S 171-264.

§ Staff Software Engineer, Modeling and Data Management Systems Section, 4800 Oak Grove Drive, M/S 171-264.

\*\* Project Scientist: University of California, Santa Cruz, University Affiliated Research Center, NASA Ames Research Center

Over the course of the past three years within the CCSDS Information Architecture Working Group (IAWG), we have been formalizing a *Reference Architecture for Space Data Systems* (or RASIM for short) whose goal is to elegantly tease apart the staunch dependency between software component and software model. RASIM was inspired by the success of the Object Oriented Data Technology (OODT) project<sup>1</sup> that originated at the Jet Propulsion Laboratory. OODT's carefully crafted reference architecture consists of a set of eight software components, and two software connectors that are instantiated and tailored for each deployment domain. Inspired by OODT, our work within CCSDS has resulted in the formalization of several core software components within RASIM including those that deal with registries and repositories, data products, archives, and query aggregation. In addition, the RASIM identifies the core data and metadata models that facilitate interoperability between information systems domains, and explores the relationships between data, metadata, and meta-metadata within space data systems. In this paper, we describe RASIM in detail, illustrate its utility through several case studies in different space data systems including NASA's Planetary Data System (PDS), and NASA's Deep Space Mission Systems (DSMS). The rest of this paper is organized as follows. Section II discusses the data and metadata models identified within RASIM. Section III discusses the software components identified within RASIM. Section IV illustrates the applicability of RASIM by applying its models to the PDS, and DSMS. Section V discusses related efforts in formalizing space data systems architecture, including those within the C3I initiative. Section VI concludes the paper.

## II. Data and Metadata Models within RASIM

RASIM identifies the *application information object*<sup>††</sup>. The application information object is the cornerstone of defining and constructing a data-driven system where models and software function in unison, but are separate entities. An application information object is an independent, flexible model of the data and corresponding metadata in an information system, and is meant to be reusable across many information system domains. The main guiding principle of the information object is to separate the models of information (e.g., data, metadata, etc.) from the actual implemented system code. In this fashion the software system and the models that describe the information in the system may both evolve independently of one another.

The information object is composed of the *data object*, a sequence of bits responsible for physically representing data, and the *metadata object*, information about the data object including, but not limited to, structure, semantic, and preservation information<sup>2</sup>.

### A. DATA OBJECTS

*Data objects* are either physical objects or digital objects as illustrated in Figure 1. A physical object is a tangible thing (e.g., a moon rock) together with some representation information bringing to light the fact that any object that can be described with data is a data object. On the other hand, a digital object is a sequence of bits, representing a thing that is not tangible (e.g., an electronic document, image file, a 'folder' of files). In contrast to the widely cited OAIS reference model<sup>3</sup>, RASIM focuses on the digital object specialization of the data object and does not focus on the physical object specialization.

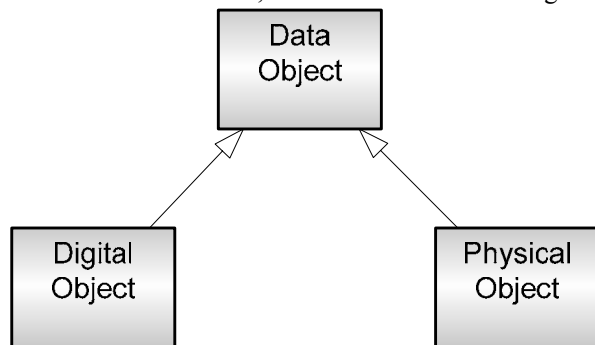
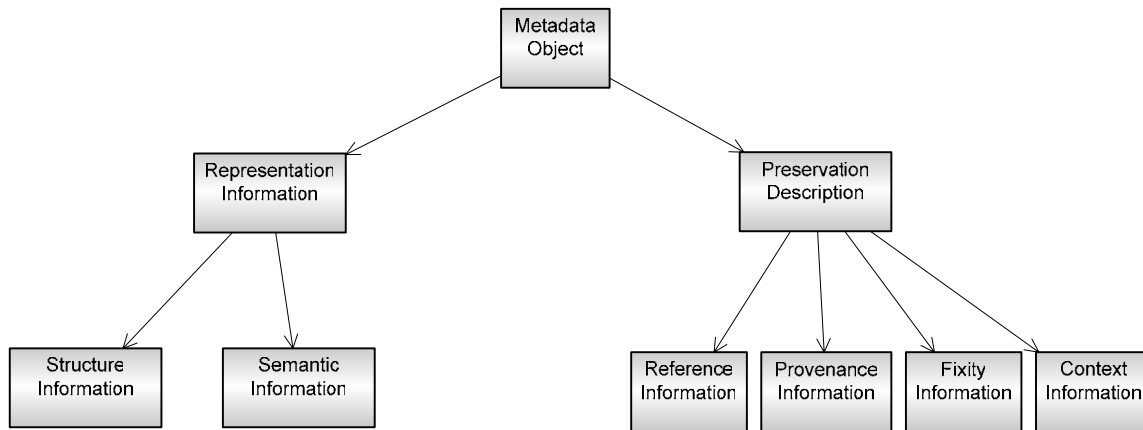


Figure 1. A Data Object

### B. METADATA OBJECTS

*Metadata objects* provide information (or *metadata*) about the data object. Similar to the OAIS reference model, a metadata object comprises representation and preservation description information as two broad classifications of metadata. As shown in Figure 2, representation information includes structure (syntactic) and semantic information and preservation information includes reference, provenance, fixity, and context information. Metadata objects might be atomic or comprised of a set of metadata sub-objects. Data objects and metadata objects are highly interdependent. Without the metadata object, essentially the data object is just a self-contained sequence of bits about which nothing is known: systems cannot unlock its information. When a

<sup>††</sup> Also used and described throughout the document as an *information object*.



**Figure 2. A Metadata Object, Adapted from reference 3.**

metadata object *and* data object are present (e.g., an information object), a myriad of capabilities are available to the user (or system). If the data object is an image, most likely the metadata object will describe what *kind* of image (JPEG or ‘raster’ for example). If the metadata object mandates that the data object has a field called *pixel*, an examination of a specified (by the metadata object) location within the data object will reveal the value of the pixel.

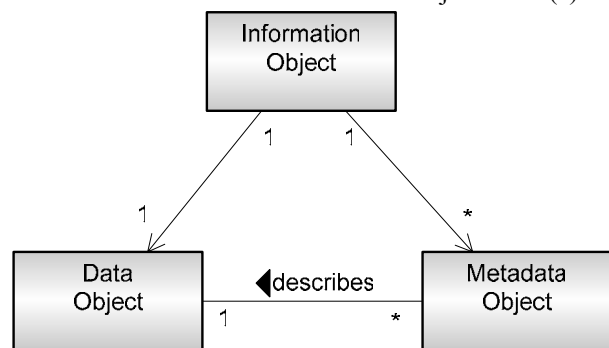
### C. INFORMATION OBJECTS

Information objects (shown in Figure 3) build upon the data and metadata object by logically associating them together. Information objects are components in information systems architecture that model both a granule of information (i.e., the *bits*) and its corresponding *metadata*. An information object consists of a data object and one or more metadata objects: the latter models the aforementioned information and metadata properties. The metadata object can describe the data object’s *structure*, such as what fields it is composed of, the fields’ *valid values* (e.g., in the case of ‘Uplink Speed’, the data may have a controlled list of available speeds such as 1MB or 2MB/sec), and the *semantic relationships* between the structural elements (such as ‘Uplink Speed **must always equal** Downlink Speed’).

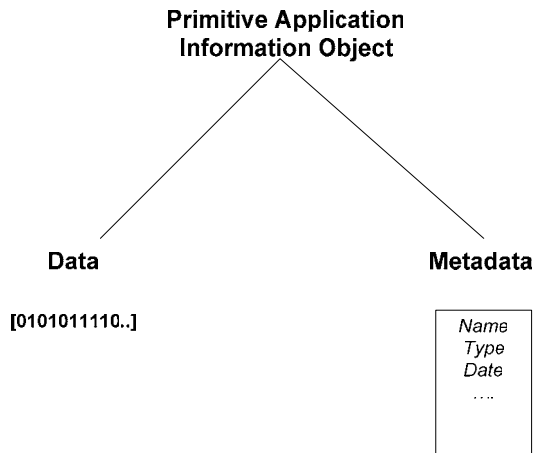
#### 1. Classes of Information Objects

There are several different classes of information objects. For brevity, in this paper we will only focus on three of the classes, however, many other classes of information objects are currently being classified and identified within the context of our CCSDS Green Book<sup>4</sup>. The three fundamental classes of information objects are: (a) the primitive information object, (b) the standard information object, and (c) the complex information object.

**Primitive Information Object** – A *primitive information object* is an information object with simple metadata information that contains a small amount of metadata with a data object. Simple metadata indicates that the only metadata captured for a particular data object are primitive attributes such as *name*, *format*, or *modification date*. These are attributes typically associated with a file in a file system and seldom provide any information about content or relationships.



**Figure 3. An information object**



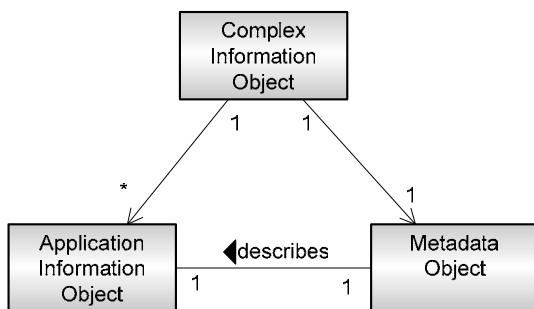
**Figure 4. Primitive Information Object Example**

An example of a primitive information object is a data file managed in a solid state recorder. Minimal metadata exists for it other than basic properties that define its name, type, and size. A name often is used to denote specialized information about an object. In practice, it is preferable to separate the name of an object from other information such as creation date, sequence numbers, etc. Many space data systems have typically focused on the management of primitive information objects and have not made metadata objects (with more complex attributes) first-class citizens

**Standard Information Object** – A *standard information object* is an information object that has well-defined metadata and a data object. The metadata is an instance of one or more domain models. The data object can be null. A number of data systems throughout the space agencies have standard information objects as part of their system design (incl. the SpaceGRID study<sup>16</sup> by ESA, and NASA’s Planetary Data

System). Standard information objects have been predominately used within archive and science processing data systems. The metadata for these information objects are often defined by some data description language like XML and may be stored in an online registry or database to enable effective search and browsing. Increasing emphasis on constructing end-to-end space information system architectures is suggesting the use of standard information objects at a variety of stages within the mission pipeline including: observation planning, execution, processing, and distribution. Standard information objects are applicable across this entire pipeline since it is a mechanism to enable interoperability between systems as long as the information objects and their associated models are carefully developed, and not intertwined with the software that supports them.

**Complex Information Object** – *Complex information objects* (shown in Figure 5) are information objects that encapsulate one or more information objects, coupled with a metadata object containing *packaging information*. Similar to the OAIS reference model, packaging information is the set of information, consisting primarily of package descriptions, which is provided to data management to support the finding, ordering, and retrieving of information holdings by consumers. Additionally packaging information is the information that is used to bind and identify the components of an information package. For example, it may be the ISO 9660 volume and directory information used on a CD-ROM to provide the content of several files containing content information and preservation description information. It also can describe the algorithms and formats of the package structure itself (e.g., whether or not the package was compressed, which compression algorithm was used, such as ZIP, TAR,<sup>##</sup> etc.).



**Figure 5. A Complex Information Object**

Each information object in a complex information object includes its own metadata that may or may not correlate with other metadata from the other information objects in the package. This makes it difficult to interpret and compare information objects, even ones that come from the same repository, unless they conform to a standard meta-model, e.g., such as the XFDU packaging model<sup>6</sup>.

The purpose of the complex information object is to provide the aggregation of related data to the user. It is assumed that the user typically knows how to use each information object within the set. If the user does not know how to correlate the information, then descriptive information related to the complex information object (such as *index information* regarding the individual information objects in the complex information object) can be used to

deduce package properties.

<sup>##</sup> See reference 5 for definitions of ZIP and TAR.

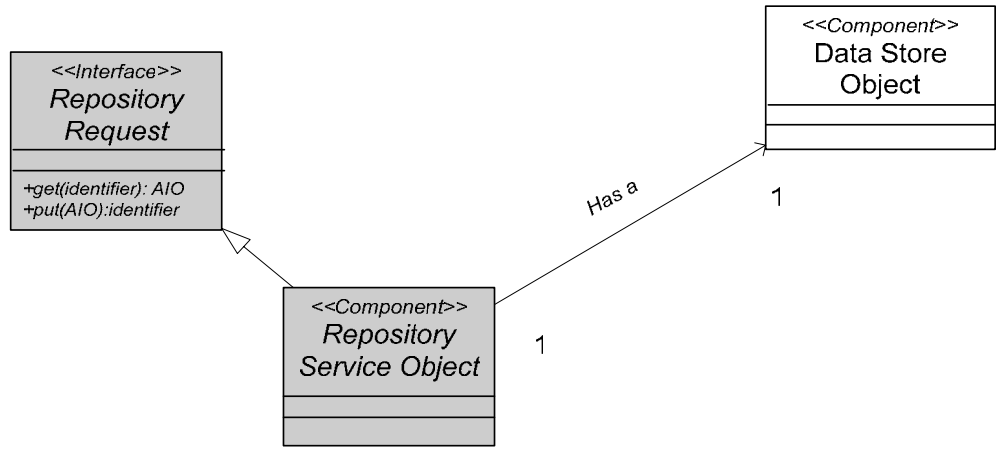


Figure 7. Repository Service Object

III. Software Components within RASIM

RASIM software components are higher level software components built using primitive data store and retrieval functions to arrive at complex capabilities. Examples of these capabilities include ingestion of data into repositories, federated search across heterogeneous repositories using registries, and the like. The set of reference components within RASIM is not meant to be comprehensive, though the RASIM set represents a sound cross-section of advanced components that span the typical usage scenarios involved in data systems. The core RASIM software components are: *Repository Service Objects*, *Registry Service Objects*, *Product Service Objects*, *Archive Service Objects*, and *Query Service Objects*.

D. REPOSITORY SERVICE OBJECT

The *repository service object* component is depicted in Figure 7. Repository service objects are responsible for management of an underlying data store object or the physical data store. The repository service object differs from a data store object by a myriad of properties that are typically considered *non-functional*. These properties include *scalability*, *dependability*, *uniformity*, and other quality attributes. In this context, repository service objects provide the same *get* and *put* methods that a typical data store object provides. However, whereas a data store object may not scale across many underlying physical data stores, may not be dependable 24x7, and may not provide a uniform software interface, a repository service object is responsible for delivering non-trivial quality of service in each of these non-functional properties.

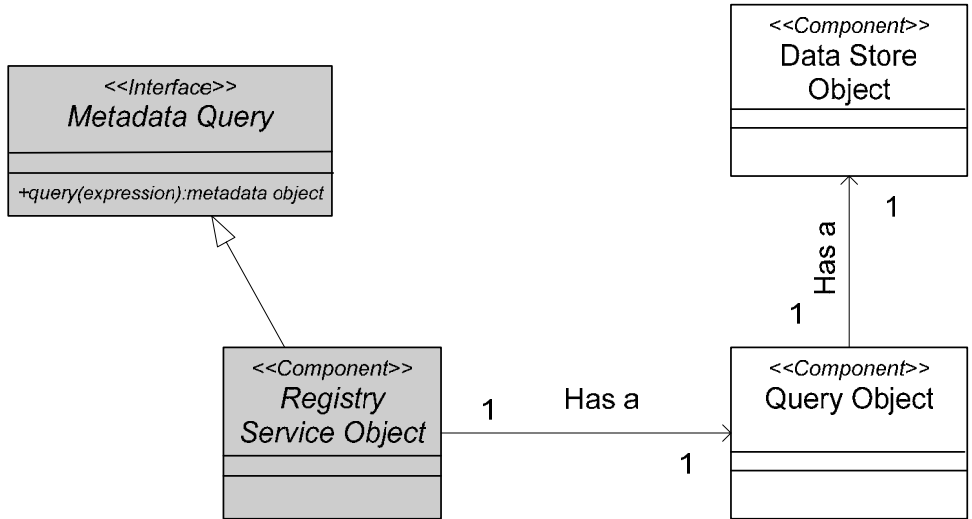


Figure 6. A Registry Service Object.

Its primary interface is a *repository request* that can be used to manage information objects. Information objects can be retrieved from the repository via the repository request interface, and a response from the repository is provided. The repository service object also provides basic *get* and *put* capabilities of information objects using the capabilities of its associated data store object.

## 2. Taxonomy of Repository Service Objects

Our work within RASIM has led us to identify several different types of repository service objects. Each of the repository service object types we have identified so far are explained below.

First, repository service objects are identified via their *type*. Type provides a quantifiable grouping for a family of repositories with similar functional and non-functional properties. We have identified three key repository types: *data store*, *operational archive*, and *long-term archive*. The *object properties* dimension serves as a general grouping of various functional and non-functional properties a repository might have. At the time of preparing this paper the properties dimension covers the entire scope of properties for a particular repository. In the long term however, properties will be categorized as dimensions of comparison and classification between different repository service objects. Potential dimensions of repositories include *compositionality*, referring to the lower-level and higher-level organization of the sub-components of a repository; *supported data objects*, referring to the type of data objects that a repository is responsible for storing; *permanence*, referring to the non-functional property of how long the data is guaranteed safe and reliable shelter within a repository; and finally *interface richness*, referring to the repository's ability to natively handle either primitive get/put operations, or higher level operations possibly requiring both querying and processing of data being returned. The last dimension in the current taxonomy, *object description*, identifies key services and responsibilities of the repository when deployed together with a set of other software components. Table 1 lists the current taxonomy and classification of repositories.

**Table 1. Taxonomy of Repository Service Objects**

Repository Object Type	Object Properties	Object Description
Data Store	Primitive Component (e.g., DBMS, and File system).	Basic Data Store component sits behind Data Store Object and supports Repository Interface to <i>get</i> and <i>put</i> data (lower level data such as streams and bits).
Operational Archive	Component that stores data products and higher level products, possibly including metadata. Supports retrieval of data products through possibly complex methods, and processing. No support for permanence. Stores products for short term (e.g., less than 10 years), and allows retrieval of products.	Advanced Component supporting retrieval of possibly complex data products, including their metadata. Repository where writes are frequent and reads are frequent. Data products stored in this type of archive will be updated and versioned. Examples of products stored in this archive are command sequence products sent using spacecraft telemetry.
Long-term Archive	Stores products for long term archiving, and supports basic archive functionality.	Archive for long-term preservation of data products, and data permanence. Supports basic archive functional interfaces (e.g., get, put).

## E. REGISTRY SERVICE OBJECT

The *registry service object* component provides an interface to retrieve metadata objects. There are two special types of metadata objects which most current registries are able to return, other than the basic metadata object aforementioned. The first type is a *service description* metadata object. A service description is some metadata document that describes the basic components of a service, such as its interface and its accepted parameters and values; a Web Services Description Language (WSDL) document<sup>7</sup> would be an example of this. The second type of

metadata object returned by most registry service objects is the *resource* metadata object. A resource metadata object is typically simple keyword-value paired information about an information object, such as an individual science data product, or a science data set. The registry service object returns metadata objects which satisfy a particular query expression provided by the user of the *metadataQuery* interface. Figure 6 depicts a registry service object.

Similar to the repository service object, there also exist different *classes* of registry service objects. A representative subset of these classes is identified below.

### 3. Taxonomy of Registry Service Objects

We have identified three main classes of registries and then classified them along a particular set of dimensions: the *registry type*, the *return object types*, and *query interface parameters*.

The three main types of registries are *metadata registry*, *service registry*, and *resource registry*. The metadata registry returns structural information describing the structure of the metadata. This is sometimes referred to as a meta-meta-model. Subsequently, the object returned from a metadata registry is a meta-metadata object. Queries to the metadata registry are formulated via specification of constraints and values assigned to a set of attributes. Constraints and values are specified either implicitly by querying the attribute's properties, or explicitly by specifying the data element's identifier.

The service registry provides an interface to search for functional services that perform a needed action specified by a user. Service registries manage descriptions of service interfaces (called *service descriptions*), including their respective locations, methods, and method parameters. New technological standards such as WSDL provide an implementation-level facility for service descriptions. An additional implementation of a service description and its respective service registry exists in the form of the *Profile Server* and *Resource Profile* components specified in references<sup>8, 9, 10</sup>. Service descriptions are important because they describe software methods, software systems, and Web resources using metadata. Because of this, they can be queried to retrieve a *service endpoint* (essentially a pointer to the service's location), and metadata describing how to invoke the particular service. This helps to facilitate the use and consumption of services dynamically via software rather than explicit invocations and requests.

The third type of registry, the resource registry, while capable of describing any resource or object, is used specifically for describing information objects such as science data products and data sets. Science catalogs such as the SIMBAD Astrophysics Catalog<sup>11</sup> are examples of resource registries that serve information objects. Resource registries can also point to other resource registries to enable discovery of information objects across distributed registries.

The classification dimensions introduced here effectively categorize the functional properties of each type of registry, leaving the non-functional classification unspecified at this point. This type of classification of non-functional registry service properties is very important, and this contribution is an element of on-going work within the Information Architecture (IA) Working Group. The taxonomy of registry service objects is summarized in Table 2.

**Table 2. A Taxonomy of Registry Service Objects**

Registry Type	Return Object Types	Query Interface Parameters
Metadata Registry	Data Dictionaries, Data Elements	Query for Data Element properties, or Data Element IDs, or Data Dictionary IDs
Service Registry	Service Endpoints, Service Metadata (interface properties, interface type, return schema)	Query for Service properties
Resource Registry	Data Products, Resource Registry Locations	Data Resource properties

## F. PRODUCT SERVICE OBJECT

The product service object contains a repository service object, coupled with a query object, and a domain processing or transformation object. The *domain processing* object is a functional component that provides

specialized processing of a data object to transform it from one object type to another. This is critical in the era of providing on-the-fly processing of data to other users and systems and allows for specialization of a core software infrastructure on a product-type specific basis. In fact, domain processing objects can be externalized and registered on a product-type basis so as to require that the object is called as part of the retrieval process. Processing can involve functions such as science level processing, compression, decompression, scaling (in the case of an image), format conversion, and many other transformations

The product service object serves as a common interface to heterogeneous data sources and allows for the querying the information objects (shown as IO in Figure 8) via a query expression. The query expression is passed along to the internal query object, which in turn evaluates the query expression and transfers it into a sequence of *get* calls to the repository service object, including execution of any specialized data processing objects. A product service object is shown in Figure 8.

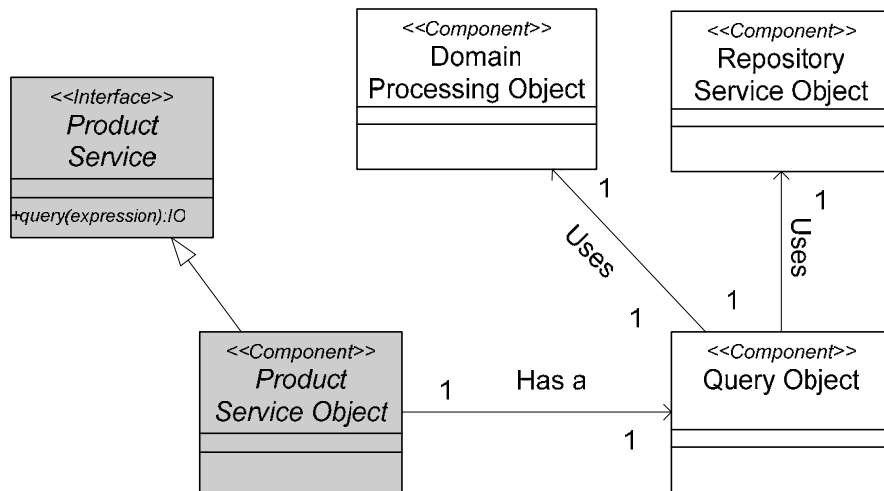


Figure 8. A Product Service Object

### G. ARCHIVE SERVICE OBJECT

Archive service objects are responsible for (a) ingestion of data objects into a repository, and (b) ingestion of metadata objects into an accompanying registry. The ingestion of both metadata and data objects can be performed using a task processing approach: the users define *tasks* formulating the ingestion process of information objects (shown as IO in Figure 9). These tasks can then be managed via a rule-based policy which, given a set of criteria such as time, task type, ingestion type, etc., determines when a particular task, or set of tasks, should be executed for a given ingestion. This rule-based task processing is often referred to as *workflow*<sup>10, 12, 13</sup> and can execute externalized objects such as the *Domain Processing Object* discussed prior. This would enable a workflow-oriented archive service to construct a pipeline for ingestion and processing of level science products from missions. The externalization of a domain processing object would allow science data processors to run on appropriate scalable hardware, such as computational clusters, constructing an architecture for science processing and archiving. In fact, this component was implemented for the SeaWinds Earth science instrument that was part of the payload for the ADEOS II satellite. This type of ingestion process is shown as the *ingest service object* component in Figure 9.

Archive service objects also have the capability of handling transaction-based ingestion of data and metadata objects, similar to the ingestion interface described in the OAIS model. This type of transaction capability would be provided by the ingest service object in Figure 9, managing all aspects of ingesting an object into the archive (e.g., validation, registration, etc.). An archive service object is shown in Figure 9.

### H. QUERY SERVICE OBJECT

The query service object manages routing of queries in order to discover and locate product service objects, repository service objects and registry service objects which contain information to satisfy user queries. Routing is accomplished by querying registry service objects in order to discover the location of the appropriate repository, or product service objects to ultimately locate the information objects (shown as IOs in Figure 10) that satisfy a user's query. Once the service objects have returned the information objects that satisfy the query, the information objects are aggregated and returned to the query service object. At that point, the query service object can perform processing such as packaging, translations to other formats, and other types of advanced processing. These advanced processing capabilities are shown as the *domain processing object* in Figure 10 and are utilized as an externalized component of the product service (recall Section III.F). Figure 10 depicts a query service object.

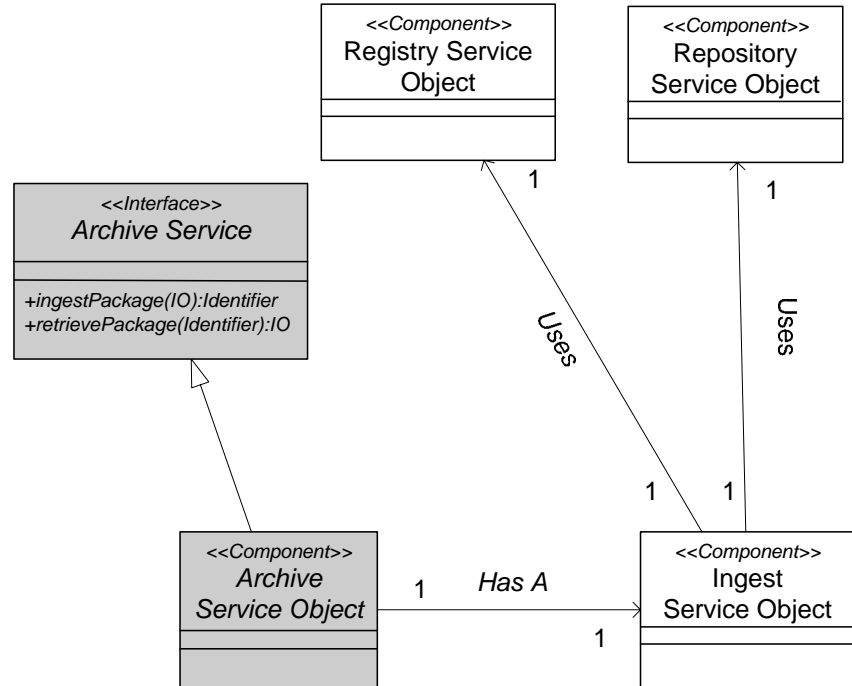


Figure 9. An Archive Service Object

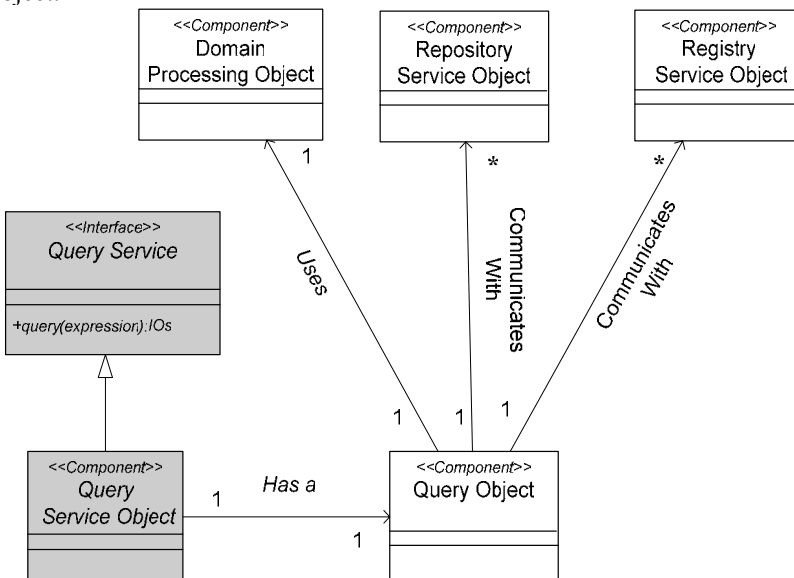


Figure 10. A Query Service Object

### IV. Applying the RASIM architecture

To demonstrate the applicability of RASIM, in this section we will model three distinct types of information objects using RASIM.

For ground data systems, a spacecraft command message file information object is discussed. For archive data systems, a planetary data system information object is discussed. Finally, for space data systems, a service link exchange (SLE) information object is discussed.

**Table 3. A Spacecraft Command Message file information object**

Data Object		Metadata Object		
<i>Name</i>	<i>Type</i>	<i>Data Element</i>	<i>Data Element Type</i>	<i>Semantic Constraints</i>
Command	Sequence of bits	Ground Station Name	String	None
		Packet-Sent Time	Timestamp	$\leq$ Current System Time
		Instrument Name	String	Value:= $a \mid a \in \{\text{spectrometer, hi-resolution imager}\}$

**I. Spacecraft Command Message File**

A spacecraft command message file is a telemetry uplink packet sent from a ground station to a spacecraft. It can be modeled using an information object. The information object is made up of a sequence of bits representing the command to be sent to the spacecraft. This bit sequence is mapped to an application information object consisting of one data object, *command sequence*. The associated structural information for the telemetry uplink packet consists of three data elements, *ground station name* (representing the ground station that sent the command to the spacecraft), *instrument name* (representing the instrument on-board the spacecraft that this sequence of commands is intended for), and *packet sent-time* (a timestamp representing the exact time the packet was sent from ground to space). Semantic information about these three data elements consists of *valid values* for the data element *instrument name* (e.g., spectrometer, or hi-resolution imager), and *min value* for the timestamp, which states that the timestamp for *packet sent-time* should be less than or equal to the current time on the sending system. This example is summarized in Table 3.

**J. Planetary Data System Product**

A Planetary Data System (PDS) product is an *archive* structure consisting of one or more science data files (e.g., image files, calibration files, SPICE files) and a PDS Label file in the ODL format. It can be represented using the information object construct. The information object consists of a set of data objects, such as *image or SPICE files*. Each data object is described by a metadata object, the PDS Label. For the SPICE files, metadata objects defined data elements such as *file name* to identify the name of the SPICE file, *Orbit Numbers* to identify the spacecraft orbit numbers that the SPICE file data covers, and *Mission Name* for which the SPICE file describes the navigation data. Each of the data elements for the SPICE files has semantic constraints. For instance, the mission name element's value must be a valid PDS mission. The orbit number element's value must be a valid orbit number from the mission. The file name of the SPICE file must exist in the PDS volume. For each the image file data objects, there are single metadata objects containing the data element *image dimensions*, which describes the width and height of

**Table 4. A Planetary Data System product information object**

Data Object		Metadata Object		
<i>Name</i>	<i>Type</i>	<i>Data Element</i>	<i>Data Element Type</i>	<i>Semantic Constraints</i>
SPICE files	Set of ancillary spacecraft data files	File Name	String	Must exist in the volume
		Orbit Numbers	Long Integer	Must be valid orbit within the mission
		Mission Name	String	Must be valid PDS Mission
Image Files	Raster Image	Image Dimensions	W x H image dimensions	Dimensions must not exceed 1024 pixels by 768 pixels

the image in pixels. There is a single semantic constraint on this element; for example, in this case, the width of the image must not exceed 1024 pixels, and the height must not exceed 768 pixels. This example is summarized in Table 4.

#### K. Space Link Extension (SLE) Service Management Objects

CCSDS is developing standards to support automation of requests between agencies for managing space link and SLE services known as ‘SLE-SM. SLE-SM defines a set of information objects called *service management objects* for automating the exchange of SLE-SM information. The service request includes the *service agreement*, *configuration profiles*, *trajectory predictions*, and *service packages*. The SLE Service Management Objects can be modeled as an information object in the same fashion shown in previous examples. The SLE Service Management information object would consist of a set of data objects including a service agreement, trajectory prediction constraints, a forward carrier agreement. Each data object would have a corresponding metadata object. In Table 5, the Trajectory Prediction Constraints data object has a metadata object associated with it that contains three data elements: *Maximum Size Storage*, of type long integer, that represents the maximum amount allowed for storage by a content manager (CM); *Allowed Trajectory Formats* is a list of acceptable CCSDS and non-CCSDS formats that this service agreement defines; and *Operation Timeout Limits* is a list of timeout values on operations involved in this service agreement. Examples of semantic constraints in the above metadata objects would be verifying that the operation timeout limits do not exceed pre-specified values, that the formats correspond to known mime type specifications, and checking to ensure that the maximum size storage does not exceed a pre-specified value.

**Table 5. SLE Service Link Exchange information object**

Data Object		Metadata Object		
<i>Name</i>	<i>Type</i>	<i>Data Element</i>	<i>Data Element Type</i>	<i>Semantic Constraints</i>
Trajectory Prediction Operations Constraints	TBD	Maximum Size Storage	Long integer	Maximum CM allowed size of storage does not exceed a pre specified value.
		Allowed trajectory formats	List	Formats conform to standard mimeType specifications.
		Operation Timeout Limits	List	Timeouts cannot exceed pre specified value.
Service Agreement	Aggregation of SLE objects	Service agreement identifier	URN	Identifier URN should be within an accepted SLE namespace

## V. Related Work

In this section, we provide information about related space data system projects which use components of RASIM. The use of information architecture components in each project is summarized in Table 6.

**Table 6. Example Projects Using Related RASIM Concepts**

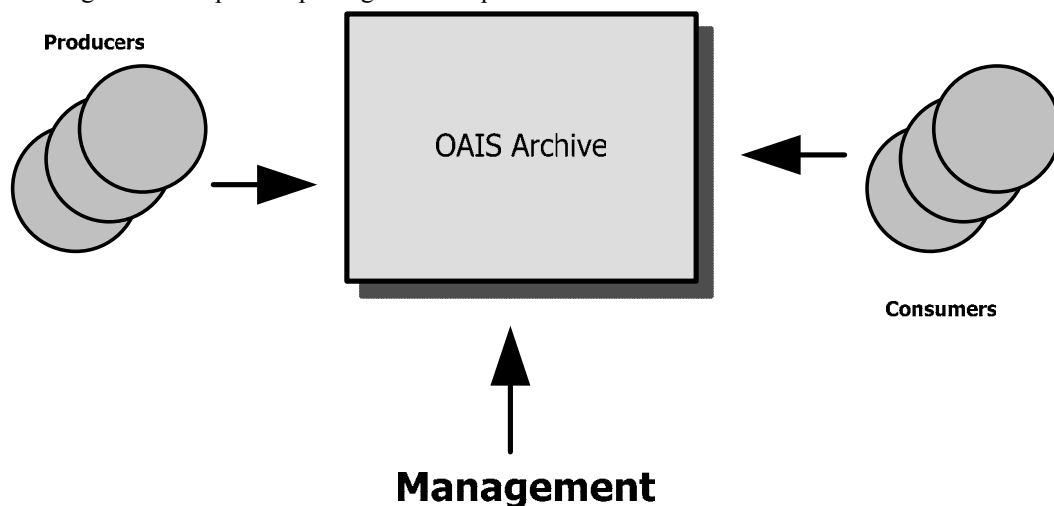
Project	Information Architecture Concepts Used
OAIS	CCSDS reference model describing information objects, information packages, archive service object.
SPACEGRID	Uses concept of information objects and registry service objects.
EOSDIS	Uses concepts including meta-models, domain models, metadata objects, information objects for a national Earth science program within NASA.
European Data Grid	Uses concept of information objects, information packages, archive service object, registry service object for a national grid system.
National Virtual Observatory	Uses concepts of information objects, information packages, archive service object, registry service object for an international astrophysics interoperability effort.
Planetary Data System	Uses concepts of information objects, information packages, archive service object, registry service object for a national planetary science grid system within NASA.

### L. OAIS

The CCSDS OAIS reference model has made metadata a key element in terms of the ability to validate ingestion of data products and understand data product format, which is a key element of RASIM. OAIS defines the notion of an ‘open archive’. An open archive is an archive service object that interacts with three main outside entities: *Producers*, *Consumers* and *Management*. In general:

- **producers** produce Submission Information Packages (SIPs) to send to the OAIS compliant archive;
- **consumers** consume Dissemination Information Packages (DIPs) that they retrieve from the OAIS compliant archive;
- **management** constitutes outside entities responsible for managing data within the archive and are not involved in the day-to-day operations of the component.

In addition to SIPs and DIPs, OAIS archives also deal with Archival Information Packages (AIPs), which are created within the OAIS archive from SIPs. Within RASIM, the OAIS DIPs, SIPs, and AIPs are each information objects conforming to each respective package format specified in the OAIS model.



**Figure 11. The Open Archival Information System Reference Model**

OAIS-compliant archives are in the business of preserving, providing, managing, and collecting information. Inherently they are domain specific implementations of the archive service object (recall Section III.G) described within RASIM.

## M. GRIDS

Recent work in the *grid* community has characterized a class of distributed data interoperable systems as *data grids*<sup>14, 15</sup>. Data grids involve the identification of (different classes of) metadata objects, used to make heterogeneous software systems interoperable. In the next paragraphs, some overviews of grid projects at various space agencies are listed. Each subsection details how the core concepts and components from RASIM are in heavy use within each of these large scale projects.

### 4. *SpaceGRID*

ESA's SpaceGRID Study<sup>16</sup> commenced in 2001 and concluded in 2003 with the goal of assessing how ESA could infuse grid technology into various Earth observing and space missions to support (1) distributed data management, (2) data distribution, (3) data access, and (4) a common architectural approach to designing, implementing, and deploying software to support such activities. The study spanned several different disciplines including Earth Observation, Space Research, Solar System Research, and Mechanical Engineering. Results of the study included identification of 240 user requirements for grids, 146 of which were considered 'common', denoting the fact that the requirement was considered useful for at least three of the study domains.

The proposed SpaceGRID infrastructure is very similar to the service objects and architectural model described in this document. It is a layered architectural model, with client applications at the top-most layer making calls through an organizational API. The organization's API makes use of grid services, which in turn use grid infrastructure to access both 'hard' (hardware-based) and 'soft' (software-based) distributed resources.

The data that is made available by grid infrastructure in the ESA report is searched using metadata catalogs. These catalogs can be thought of as storing metadata objects, which in turn point to data objects desired by the user. Effectively, the grid infrastructure described in the SpaceGRID report is distributing, searching, and delivering information objects to users.

### 5. *EOSDIS*

NASA's Earth Observing System Data and Information System, or EOSDIS, was a preliminary investigation into how NASA could support data distribution, processing, archival and storage of Earth science data sets produced by Earth observing missions. EOSDIS was an excellent early example of the problems with state-of-the-art information systems technology circa 1996. So-called "one-off" data systems were being produced across the country, and viable data sets could not be accessed, distributed and ultimately used. This required sending data on removable media and ultimately increased the amount of time necessary to engage in science. The goal of EOSDIS was to bridge the gap between existing Earth science data systems, and unlock their data, and make it available to scientists.

Many of the conclusions from EOSDIS were early precursors to the study and ultimate adoption and acceptance of the *grid paradigm*. The relation between EOSDIS and RASIM lies in the fact that EOSDIS is a domain-specific example of (1) Earth science specific information objects, (2) Earth science meta-models, (3) Earth science metadata objects, and (4) Earth science domain models and ontologies.

### 6. *European Data Grid*

The European Data Grid (EDG) is an EU- and ESA-funded project aimed at enabling access to geographically distributed data and computational resources<sup>17</sup>. EDG uses Globus Toolkit technology to support base grid infrastructure and then builds data-specific services on top of the underlying grid infrastructure. These data-specific services are services such as *replica management*, *metadata management*, and *storage management*. Because of its focus on data and metadata, EDG is highly related to RASIM. The EDG system manages, distributes, processes, and archives information objects. The metadata objects are stored in metadata catalogs, and the data objects are stored transparently in an underlying storage system. Users use software components, similar to those described in section **Error! Reference source not found.**, to query for and retrieve application information objects and information packages made available by the EDG system.

### 7. *National Virtual Observatory*

The National Virtual Observatory, or NVO, is an NSF-funded project whose goal is to enable science by greatly enhancing access to data and computational resources. NVO uses the Globus Toolkit grid middleware infrastructure

to distribute, process, retrieve, and search for astrophysical science data. The shared components within NVO and RASIM are: (1) a well defined component architecture, including information objects (or astrophysical data products), (2) common models to describe the information objects, and (3) software service objects (in the form of grid services) to exchange science data.

#### 8. *The C3I Initiative*

The Command, Control, Communications and Information Architecture (C3I) initiative is formulating an information architecture for Constellation. This project encompasses the next generation shuttle and accompanying assets. This project will see an order of magnitude increase in assets and complexity which will require a focus on architecture in order to mitigate risk. Current missions require a team of personnel to coordinate the command, control, and communication for a single asset. It is not reasonable to assume that such a team would be deployed for every asset associated with Constellation, as the increase in personnel and coordination effort would overwhelm the budget and produce a quagmire of effort. Therefore, it is evident that traditional ground and space support will not suffice for this next generation command, control, and communication systems. Furthermore, it is clear that efforts such as the CCSDS IAWG can be leveraged to architect an information architecture that will scale to the foreseen requirements.

## VI. Conclusion

In this paper we presented the Reference Architecture for Space Information Management (or RASIM) and illustrated how RASIM stresses the importance of separating software that manages information from the models of the information being managed. We discussed the core functional software components defined by RASIM, and the information models managed by those software components. We explored the architecture's utility by providing examples of RASIM's usage within existing space data systems and efforts.

## Acknowledgments

This effort was supported by the Jet Propulsion Laboratory, managed by the California Institute of Technology. R. Roshandel provided valuable input into earlier versions of this paper. The authors would like to acknowledge the contributions of the CCSDS Information Architecture Working Group, who provided helpful suggestions and input into material used in this document. The authors would like to acknowledge the financial support of Peter Shames and the JPL Data Standards program.

## References

1. Mattmann, C., Crichton, D., Medvidovic, N., and Hughes, J. S. A Software Architectural Framework for Highly Distributed and Data-Intensive Scientific Applications. *Proc. of the 28<sup>th</sup> International Conference on Software Engineering*, Shanghai, China, May 2006.
2. The Data Description Language EAST Specification (CCSD0010). Recommendation for Space Data System Standards, CCSDS 644.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, November 2000.
3. *Reference Model for an Open Archival Information System (OAIS)*. Recommendation for Space Data System Standards, CCSDS 650.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, January 2002.
4. *Information Architecture Reference Model*. Draft Informational Report, CCSDS-312.0-G-0. Draft Green Book – Issue 1. Washington D.C., CCSDS, March 2006.
5. Hyperdictionary. <http://hyperdictionary.com/>
6. Lou Reich. “XML Packaging for the Archiving and exchange of Binary Data and Metadata.” In *Proceedings of the 2003 Open Forum on Metadata Registries (Santa Fe, New Mexico)*. 2003. <http://metadata-standards.org/>
7. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. Web Services Description Language (WSDL) 1.1. W3C Recommendation, 2001, <http://www.w3.org/TR/wsdl>
8. Crichton, D., et al. A Component Framework Supporting Peer Services for Space Data Management. In *Proc. of the 2002 IEEE Aerospace Conference* (Big Sky, Montana), 2639-2649. Piscataway, NJ: IEEE, 2002.
9. Crichton, D., Hughes, J.S. and Kelly, S. A Science Data System Architecture for Information Retrieval. In *Clustering and Information Retrieval*, edited by W. Wu, H. Xiong, and S. Shekhar, 261-298. Network Theory and Applications. Norwell, Massachusetts, USA, and Dordrecht, The Netherlands: Kluwer, 2003.
10. Deelman, E et al. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing* 1, no. 1 (2003): 9-23.
11. The SIMBAD Astronomical Database. Centre de Données Astronomiques de Strasbourg. <http://cdsweb.u-strasbg.fr/Simbad.html>

12. Blythe, J., Deelman, E., and Gil, Y. Automatically Composed Workflows for Grid Environments. *IEEE Intelligent Systems* 19, no. 4 (July/August 2004): 16-23
13. Deelman, E., et al. Grid-Based Galaxy Morphology Analysis for the National Virtual Observatory. In *Proc.s of the 2003 IEEE Conference on Supercomputing* (Phoenix, AZ). Los Alamitos, CA, USA: IEEE Computer Society, 2003.
14. Chervenak, A. et al.. A Framework for Constructing Scalable Replica Location Services. In *Proc.s of the 2002 ACM/IEEE Conference on Supercomputing* (Baltimore, Maryland), 1-17. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002.
15. Chervenak, A., et al. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications* 23 (2001): 187-200.
16. *SpaceGRID Study Final Report*. SGD-SYS-DAT-TN-100-1.2. Issue 1.2. SpaceGRID Consortium, 2003.
17. Roberto Puccinelli. "An Introduction to DataGrid." Illustrated by Aldo Stentella. March 2004. The DataGrid Project. <<http://web.datagrid.cnr.it/LearnMore/index.jsp>>