The Consultative Committee for Space Data Systems

Report Concerning Space Data System Standards

REFERENCE ARCHITECTURE FOR SPACE INFORMATION MANAGEMENT

INFORMATIONAL REPORT

CCSDS 312.0-G-1

GREEN BOOK

March 2013

The Consultative Committee for Space Data Systems

Report Concerning Space Data System Standards

REFERENCE ARCHITECTURE FOR SPACE INFORMATION MANAGEMENT

INFORMATIONAL REPORT

CCSDS 312.0-G-1

GREEN BOOK

March 2013

AUTHORITY

Issue:	Green Book, Issue 1
Date:	March 2013
Location:	Washington, DC, USA

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and reflects the consensus of technical working group experts from CCSDS Member Agencies. The procedure for review and authorization of CCSDS Reports is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems*.

This document is published and maintained by:

CCSDS Secretariat
Space Communications and Navigation Office, 7L70
Space Operations Mission Directorate
NASA Headquarters
Washington, DC 20546-0001, USA

FOREWORD

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Report is therefore subject to CCSDS document management and change control procedures, which are defined in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-3). Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be addressed to the CCSDS Secretariat at the address indicated on page i.

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

Member Agencies

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

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Federal Science Policy Office (BFSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- Chinese Academy of Sciences (CAS)/China.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- CSIR Satellite Applications Centre (CSIR)/Republic of South Africa.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 312.0-G-1	Reference Architecture for Space Information Management, Informational Report, Issue 1	March 2013	Current issue

CONTENTS

Section Page

1 INTRODUCTION 1-1

1.1 PURPOSE 1-1
1.2 SCOPE AND APPLICABILITY 1-3
1.3 TERMINOLOGY 1-3
1.4 REFERENCES 1-4

2 INFORMATION ARCHITECTURE 2-1

2.1 INTRODUCTION 2-1
2.2 INFORMATION OBJECTS 2-4
2.3 MODELING CONCEPTS 2-11
2.4 INTEROPERABILITY 2-16

3 SOFTWARE COMPONENTS FOR INFORMATION ARCHITECTURE 3-1

3.1 OVERVIEW 3-1
3.2 PRIMITIVE INFORMATION MANAGEMENT OBJECTS 3-1
3.3 ADVANCED INFORMATION MANAGEMENT OBJECTS 3-5

4 DISTRIBUTED INFORMATION SERVICE ARCHITECTURES 4-1

4.1 INFORMATION SERVICE ARCHITECTURE VIEWS 4-2
4.2 SERVICES AND PATTERNS 4-3
4.3 INTEGRATED INFORMATION SERVICE ARCHITECTURE 4-8
4.4 INFORMATION MODEL-DRIVEN ARCHITECTURES 4-11

5 SPACE DATA SYSTEMS PROJECTS 5-1

5.1 OVERVIEW 5-1
5.2 OPEN ARCHIVAL INFORMATION SYSTEM 5-1
5.3 GRIDS 5-2

6 RELATIONSHIP OF THIS DOCUMENT TO OTHER CCSDS STANDARDS EFFORTS 6-1

ANNEX A APPLICABLE STANDARDS USED IN THIS DOCUMENT A-1
ANNEX B ABBREVIATIONS AND ACRONYMS B-1

CONTENTS (continued)

Figure Page

1-1 High-Level Abstract View of Interoperable Information Architecture 1-2
2-1 A Data Object 2-3
2-2 A Metadata Object, Adapted from Reference [5] 2-4
2-3 An Information Object 2-4
2-4 Primitive Information Object Example 2-5
2-5 A Complex Information Object 2-6
2-6 Service Agreement Information Model Overview 2-10
2-7 Information Object in Context 2-12
2-8 Model Hierarchy, Adapted from Reference [24] 2-12
2-9 Planetary Science Domain Model Example (Simplified) 2-14
2-10 Data Models, Meta-Models, and Domains 2-17
3-1 The Internal Structure of a Physical Data Storage 3-2
3-2 A Data Store Object 3-3
3-3 The *Put* Operation of the Data Store Object 3-3
3-4 The *Get* Operation of the Data Store Object 3-4
3-5 A Query Object 3-4
3-6 The *Find* Operation of the Query Object 3-5
3-7 Repository Service Object 3-6
3-8 A Registry Service Object 3-8
3-9 A Product Service Object 3-10
3-10 An Archive Service Object 3-11
3-11 A Query Service Object 3-12
4-1 Service Description Conceptual Model 4-3
4-2 Static View (Left) and Runtime View (Right) of an Information Service Architecture 4-4
4-3 Information Services Connected to a Software Bus 4-6
4-4 REST-Based Distributed Architecture with Information Services on the Back-End 4-7
4-5 Space Domain Functions and Information Services in a Distributed Environment 4-9
4-6 Information Service View of the CCSDS Layered Stack 4-10
4-7 Relationship between Information Architecture Concepts 4-12
4-8 Relationship between Information Models and Services 4-13
5-1 The OAIS Reference Model, Adapted from Reference [5] 5-2
5-2 SpaceGRID Proposed Infrastructure 5-4

Table

2-1 Information Object View of a Spacecraft Command Message File 2-8
2-2 Information Object View of a Planetary Data System Product 2-9
2-3 Information Object View of an SLE Service Management Object 2-11
3-1 A Taxonomy of Repository Service Objects 3-7
3-2 A Taxonomy of Registry Service Objects 3-9
5-1 Example Projects Using Related RASIM Concepts 5-1
A-1 CCSDS Information Standards Mapped to Information Architecture Concept A-3

1 INTRODUCTION

1.1 PURPOSE

In the absence of sufficient information system standards for interoperability and cross support, systems have been developed that do not allow the exchange of information across the ground and flight side of information systems. Often these systems do not allow for integrated exchange of information between components within these environments, let alone among space agencies. The lack of any type of architectural model or standard services for the discovery, access, management, and transformation of information has led to non-interoperable systems.

The focus of this document is to present a reference space information management architecture (information architecture) that encompasses the capture, management, access, and exchange of data for both flight and ground systems across the operational mission lifecycle. This includes identification of a set of conceptual functional components for information management, definition of their interfaces for information management, representation of these components and interfaces, and definitions of information management processes (interactions between users and systems).

The intent of this document is to provide a conceptual basis on which standards can be developed to support information management across the entire mission environment. This document defines the necessary concepts and terminology for information architecture and leverages much of the past CCSDS and data management community work in this area. Part of this leveraging includes defining how existing standards can be assembled to fit into an information architecture for deploying space data systems. The information architecture covers problem areas associated with space data systems (such as organizational, functional, operational, and cross-support issues).

The information architecture presented within this document is layered. To achieve interoperability both within and across domains, and across applications built based on this document, each layer should be addressed. Figure 1-1 depicts this view and identifies one possible means of achieving interoperability at each layer. Each layer is critical to achieving interoperability. At the software and data levels, in order to achieve system interoperability, it is essential that common interfaces and meta-models for the content of messages flowing between application interfaces be defined along with common definitions for the data itself. This architecture document purposely separates into sections the information architecture and the information management components that implement that architecture. The separation of the information architecture from the software architecture promotes reuse, provided that the software components can be configured by common meta-models.

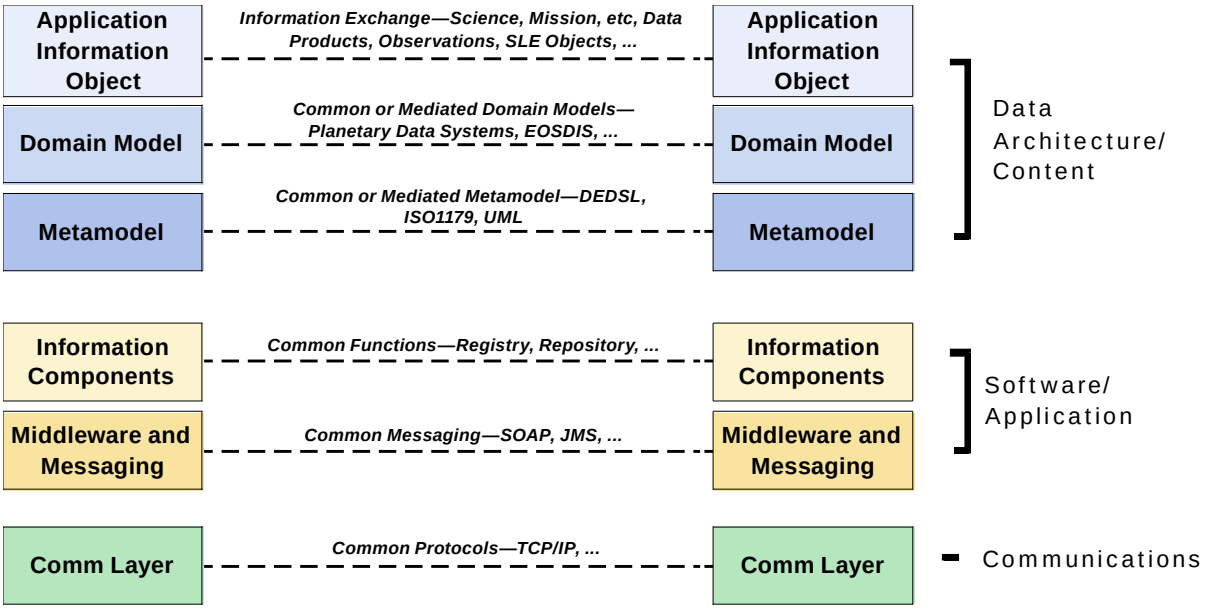


Figure 1-1: High-Level Abstract View of Interoperable Information Architecture

The concept of having multi-mission, but common meta-models is critical to achieving multi-mission, cross-agency interoperability. For example, a common eXtensible Markup Language (XML) schema defined to annotate telemetry data files could be developed to support improved information management of telemetry systems. Structuring the XML document in such a way as to enable a common cataloging function (to catalog the metadata in the XML document across missions) would provide a multi-mission capability. If this XML schema is derived from a core meta-model, then it could also support annotation of other data objects, such as science data objects. This would enable the *same cataloging function*, deployed in a *completely different* part of the ground system, to catalog the science data. Architecting systems to consider the underlying models, how they are derived, and how they can be used by a core set of information components, will increase the longevity of software systems design and promote an infrastructure which enables improved utility of the data generated from international space missions.

The document is structured into sections addressing key elements of information management architecture: data and information architecture; software components for information architecture; and distributed information services architectures that can be constructed using the approaches. The last section describes some existing space data system projects that already utilize, or could benefit from, the use of the information architecture described in this document. Figure 1-2 is a class diagram showing the relationships among these key elements.

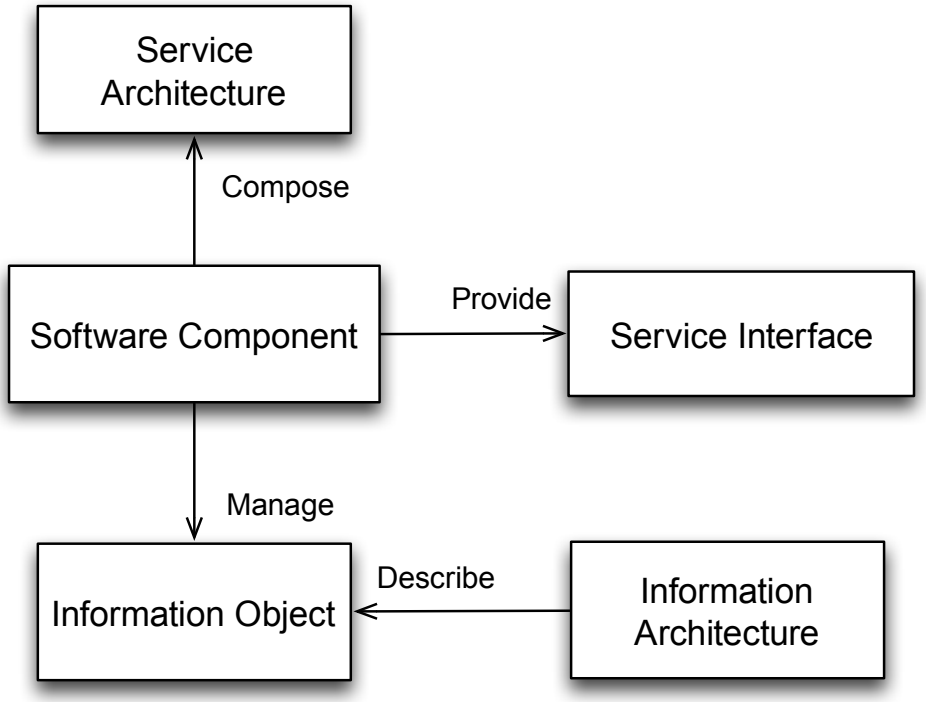


Figure 1-2: Class Diagram of Information Management Architecture Elements

1.2 SCOPE AND APPLICABILITY

This document is intended to provide an overview and background for those interested in understanding and developing information architectural elements for building space data systems. These elements include software components, such as registries and repositories, and data components and interfaces. Mission operation ground data systems, science data systems, and onboard data management systems can all benefit from applying these architectural principles and concepts, but this document does not provide specific approaches for any of these possible specific applications. This document is most applicable in complex environments such as space, but clearly has the potential to provide a roadmap for information architecture in many types of data management systems.

NOTE – This document is provided for informational purposes only. While it provides a useful set of concepts and terminology, the working group that produced it was terminated due to a lack of agency resources, and there is no companion standard or recommended practice. As such this document is not binding upon other CCSDS standards or working groups.

1.3 TERMINOLOGY

The following terminology is used throughout the document.

model: A model provides a specification for representing objects and their relationships.

metadata: Metadata is literally 'data about data', i.e., information that describes another set of data.

meta-model: A meta-model is a model which describes another model.

schema: A schema is a means for defining the structure, content, and, to some extent, the semantics of data.

application information object: An application information object (AIO) is an object containing an internal data object and a metadata object.

application information architecture: An application information architecture is the notion of architecting information systems across system domains (e.g., space data systems, archiving systems, biomedical informatics systems) with a focus on both data architecture and software architectural concerns.

data architecture: A data architecture is the specification of the overall structure, logical components, and the logical interrelationships of data and information.

software architecture: A software architecture is the specification of the overall structure, behavior, logical components, and logical interrelationships of a software system.

data product: A data product is the result of an active function which produces data. A data product may be simple and include just data value, or it may be complex and contain both data and metadata objects.

1.4 REFERENCES

[1] "Association, Aggregation and Composition." June 1998. Object Orientation Tips. http://ootips.org/uml-hasa.html.

[2] Mandar Chitnis, Pravin Tiwari, and Lakshmi Ananthamurthy. "The UML Class Diagram: Part 1." May 2003. Developer.com. http://www.developer.com/design/article.php/2206791.

[3] "The SIMBAD Astronomical Database." Centre de Données Astronomiques de Strasbourg. http://cdsweb.u-strasbg.fr/Simbad.html.

[4] J. Blythe, E. Deelman, and Y. Gil. "Automatically Composed Workflows for Grid Environments." *IEEE Intelligent Systems* 19, no. 4 (July/August 2004): 16-23.

[5] *Reference Model for an Open Archival Information System (OAIS).* Recommendation for Space Data System Practices, CCSDS 650.0-M-2. Magenta Book. Issue 2. Washington, D.C.: CCSDS, June 2012.

[6] *The Data Description Language EAST Specification (CCSD0010)*. Recommendation for Space Data System Standards, CCSDS 644.0-B-3. Blue Book. Issue 3. Washington, D.C.: CCSDS, June 2010.

[7] *Data Entity Dictionary Specification Language (DEDSL)—XML/DTD Syntax (CCSD0013)*. Recommendation for Space Data System Standards, CCSDS 647.3-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, January 2002.

[8] Ann Chervenak, et al. "A Framework for Constructing Scalable Replica Location Services." In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing (Baltimore, Maryland)*, 1–17. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002.

[9] Ann Chervenak, 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.

[10] D. Crichton, et al. "A Component Framework Supporting Peer Services for Space Data Management." In *Proceedings of the 2002 IEEE Aerospace Conference (Big Sky, Montana)*, 2639–2649. Piscataway, NJ: IEEE, 2002.

[11] D. J. Crichton, J. S. Hughes, and S. Kelly. "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.

[12] Ewa Deelman, et al. "Mapping Abstract Complex Workflows onto Grid Environments." *Journal of Grid Computing* 1, no. 1 (2003): 9–23.

[13] Ewa Deelman, et al. "Grid-Based Galaxy Morphology Analysis for the National Virtual Observatory." In *Proceedings of the 2003 IEEE Conference on Supercomputing (Phoenix, AZ)*. Los Alamitos, CA, USA: IEEE Computer Society, 2003.

[14] "EOSDIS Core System Data Model." EOSDIS - Earth Data Website. http://earthdata.nasa.gov/our-community/esdswg/standards-process-spg/heritage-standards/eosdis-core-system-data-model.

[15] Roberto Puccinelli. "An Introduction to DataGrid." Illustrated by Aldo Stentella. March 2004. The DataGrid Project. http://web.datagrid.cnr.it/LearnMore/index.jsp.

[16] The Globus Alliance. The University of Chicago/Argonne National Laboratory. http://www.globus.org/.

[17] H. Gomaa, D. Menasc, and L. Kerschberg. "A Software Architectural Design Method for Large-Scale Distributed Information Systems." *Distributed Systems Engineering* 3, no. 3 (1996): 162-172.

[18] Hyperdictionary. http://hyperdictionary.com/.

[19] *Information Technology—Metadata Registries (MDR)—Part 1: Framework*. International Standard, ISO/IEC 11179-1:2004. 2nd ed. Geneva: ISO, 2004.

[20] C. Kesselman, I. Foster, and S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations." *The International Journal of High Performance Computing Applications* 15, no. 3 (Fall 2001): 200-222.

[21] *SpaceGRID Study Final Report*. Issue 1.2. SGD-SYS-DAT-TN-100-1.2. ESA Contract No. 15369/01/I-LG. N.p.: SpaceGRID Consortium, September 2003.

[22] Chris A. Mattmann, et al. "Software Architecture for Large-Scale, Distributed, Data-Intensive Systems." In *Proceedings of the 4th IEEE/IFIP Working Conference on Software Architecture (WICSA-4, Oslo, Norway)*, 255-264. Los Alamitos, CA, USA: IEEE Computer Society Press, 2004.

[23] R.W. Moore, et al. "Data-Intensive Computing." In *The Grid: Blueprint for a New Computing Infrastructure*, edited by Ian Foster and Carl Kesselman, 105-130. San Francisco: Morgan Kaufmann Publishers, 1999.

[24] Object Management Group. http://www.omg.org/.

[25] 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)*. Washington, D.C.: ISO/IEC JTC1 SC32 WG2, 2003.

[26] Gurmeet Singh, et al. "A Metadata Catalog Service for Data Intensive Applications." In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing (Phoenix, AZ)*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2003.

[27] SPASE Consortium. *A Space and Solar Physics Data Model*. Version 1.0.1. N.p.: SPASE Consortium, January 2006.

[28] Tim Bray, et al., eds. *Extensible Markup Language (XML) 1.0*. 5th ed. W3C Recommendation. N.p.: W3C, November 2008.

[29] *XML Formatted Data Unit (XFDU) Structure and Construction Rules*. Recommendation for Space Data System Standards, CCSDS 661.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, September 2008.

[30] *Parameter Value Language Specification (CCSD0006 and CCSD0008)*. Recommendation for Space Data System Standards, CCSDS 641.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, June 2000.

[31] Pedro Osuna, et al. *NASA-PDS/ESA-PSA Planetary Data Interoperability*. White Paper. N.p.: NASA/ESA, July 2005.

[32] *Systems and Software Engineering—Architecture Description*. International Standard, ISO/IEC/IEEE 42010:2011. Geneva: ISO, 2011.

[33] C. Matthew MacKenzie, et al., eds. *Reference Model for Service Oriented Architecture 1.0*. OASIS Standard, 12 October 2006. Burlington, Massachusetts: OASIS, 2006.

[34] *Reference Architecture for Space Data Systems*. Recommendation for Space Data System Practices, CCSDS 311.0-M-1. Magenta Book. Issue 1. Washington, D.C.: CCSDS, September 2008.

[35] *Space Communication Cross Support—Service Management—Service Specification.* Recommendation for Space Data System Standards, CCSDS 910.11-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, August 2009.

2 INFORMATION ARCHITECTURE

2.1 INTRODUCTION

2.1.1 GENEREAL

Software systems of today are growing in complexity, dynamicity, and heterogeneity, and are becoming increasingly more costly to operate. Space data systems are a representative example of this emerging trend. Data systems in the space systems application domain are highly distributed, complex software entities that must manage information end-to-end during mission operations. These 'ends' include mission planning, scheduling and command directives, observation by a scientific instrument, downlink via one of many existing CCSDS protocols to a ground station on Earth, delivery to a science processing center, distribution to the scientific community, and ultimately distribution to an archive center for preservation. Because of the complexity of such distributed entities, which are often built by different organizations at different times and for somewhat different purposes, the space data system needs to be driven by the *model*s of the information that it will process, distribute, and manage. This means models of an image on a spacecraft. It means models of engineering data that need to be sent to a control operations center. It also means *models* of other *models*. There are many different models that need to be managed across an end-to-end space data system. To avoid rigidity, however, software used by a space data system should be flexible: it should be driven by the models on which it operates, and not vice versa.

For the most part, however, current space data systems are not flexible, and often include software implementations that *are* extremely tied to the information on which they operate. As systems become increasingly distributed, this problem becomes worse. A well-defined architecture for managing information becomes critical in order to decouple software and information architectures and provide common mechanisms for systems to be interoperable. While space data systems serve as a prime example of this challenge, other domains have similar challenges as well. Science processing systems and space flight operation systems all exhibit the same structure: 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.

In this document, the *application information object*[1] is described. 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. Modularity, separation of concerns, and dynamic evolution of information system components are only a representative cross-section of the benefits that this model provides.

[1] Also used and described throughout the document as an *information object*.

The information object is composed of a *data object*—a sequence of bits responsible for physically representing data, and a *metadata object*—information about the data object including, but not limited to, structure, semantic, and preservation information (reference [6]). This section starts by providing key definitions and is followed by a small taxonomy of information object types commonly used in information architecture and a set of Standard Information Object examples across domains for clarification. The section concludes with definitions of *meta-models, domain models, and data dictionaries,* which play a key role in the description of information objects. Figure 2-1 shows the relationships among these different types of information objects.

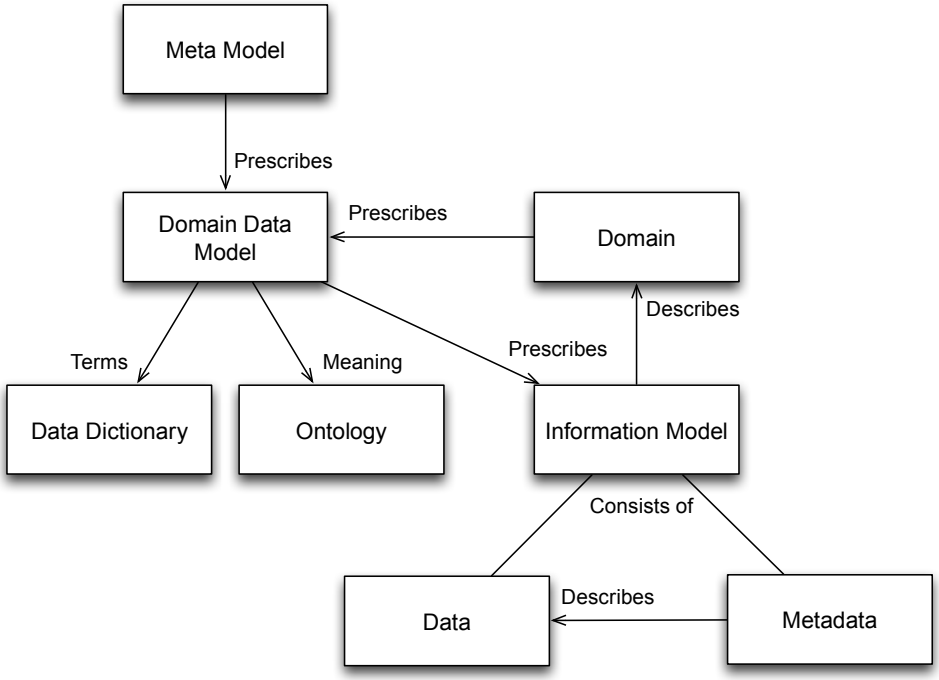


Figure 2-1: Class Diagrams of Information Objects

2.1.2 DATA OBJECTS

Data objects are either physical objects or digital objects as illustrated in figure 2-2. A physical object is a tangible thing (e.g., a moon rock) together with some representation information; this brings 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). This document focuses on the digital object specialization of the data object; the physical object specialization is not considered.

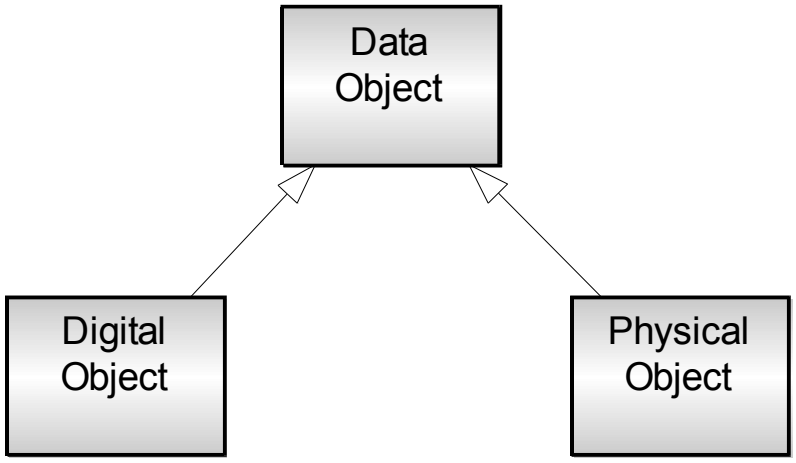


Figure 2-2: A Data Object

2.1.3 METADATA OBJECTS

Metadata objects in this document provide information (or *metadata*) about the data object. Similar to the OAIS reference model (reference [5]), a metadata object in this document comprises representation and preservation description information as two broad classifications of metadata. As shown in figure 2-3**,** representation information includes structure (syntactic) and semantic information, and preservation information includes reference, provenance, fixity, and context information. Also, the metadata objects described in this document might be atomic or comprised of a set of metadata sub-objects. Data objects and metadata objects are highly interdependent. Without the metadata object, the data object is just a self-contained sequence of bits about which nothing is known; systems cannot unlock its information by simple examination of the object. When a metadata object *and* data object are present (e.g., an information object), the user (or system) is able to derive information from it. 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 location within the data object (as defined by the metadata object) will reveal the value of the pixel.

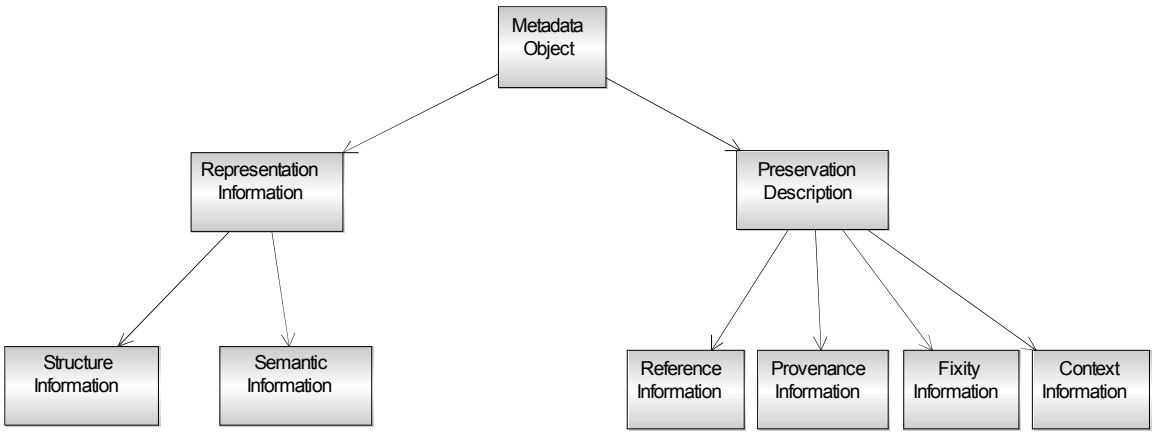


Figure 2-3: A Metadata Object, Adapted from Reference [5]

2.2 INFORMATION OBJECTS

2.2.1 GENERAL

Information objects (shown in figure 2-4) build upon the data and metadata objects by logically associating them. Information objects are components in information architecture that model both a granule of information (i.e., the *bits*) and its corresponding *metadata*. An information object consists of a data object plus 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 be equal or less than** Downlink Speed').

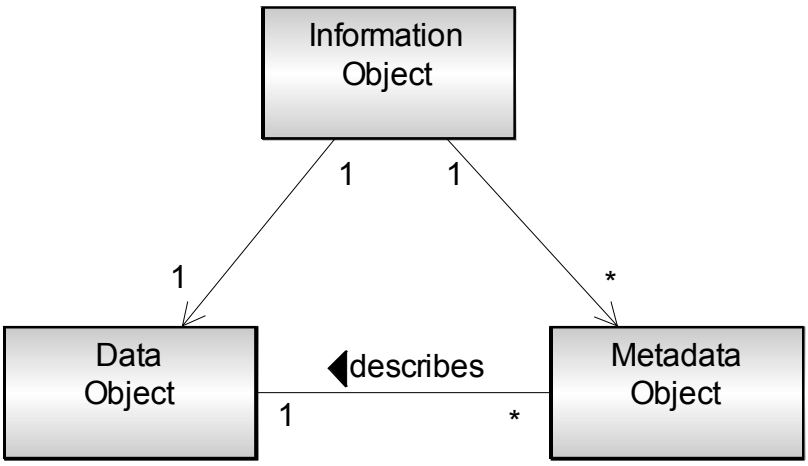


Figure 2-4: An Information Object

2.2.2 TAXONOMY OF INFORMATION OBJECTS

2.2.2.1 Overview

For the purposes of comparing different information objects, this subsection identifies a set of information object *classes*. They are detailed below.

2.2.2.2 Primitive Information Object

As shown in figure 2-5, 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, and modification date. These are attributes typically associated with a file that is in a file system; the attributes seldom provide any information about content or relationships.

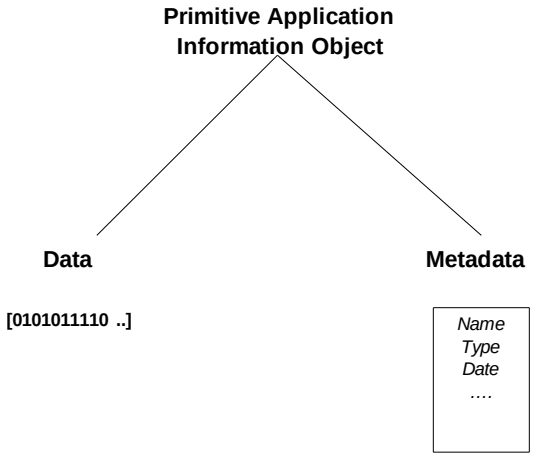


Figure 2-5: Primitive Information Object Example

An example of a primitive information object is a data file managed in a solid-state recorder. No metadata may exist for it other than basic properties that define its name, type, size, and time of acquisition. 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 typically have focused on the management of primitive information objects, used file names to capture some limited metadata, and have not made metadata objects first-class citizens.

2.2.2.3 Standard Information Object

A *standard information object* is defined as 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. These 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 mission information system architectures will require that standard information objects be used at a variety of stages including observation planning, execution, processing, and distribution across the mission pipeline. 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 planned and used consistently.

2.2.2.4 Complex Information Object

Complex information objects (shown in figure 2-6) are information objects that encapsulate one or more information objects, coupled with a metadata object containing *packaging information*. Similar to the OAIS reference model (reference [5]), 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,[2] etc.).

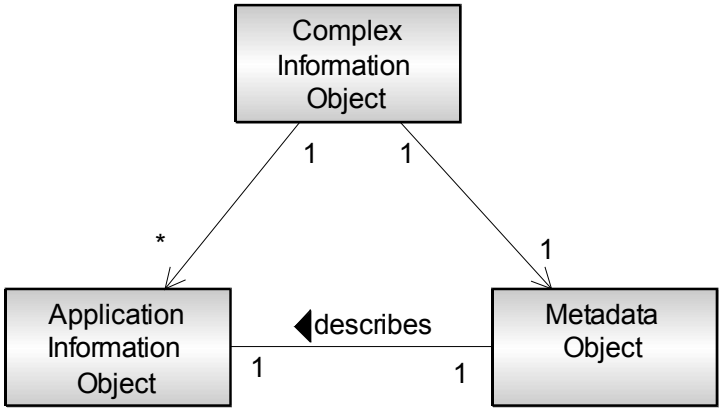


Figure 2-6: A Complex Information Object

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

[2] See reference [18] for definitions of ZIP and TAR.

from the same repository, unless they conform to a standard meta-model, e.g., the XFDU packaging model (reference [25]).

The purpose of the complex information object is to provide an aggregated set 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.

2.2.3 EXAMPLES OF INFORMATION OBJECTS

2.2.3.1 Overview

This subsection explores information objects through several illustrative examples in the context of different application domains. 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 Space Link Extension (SLE) information object is discussed.

2.2.3.2 Spacecraft Command Message File

A spacecraft command message file is a set of telemetry uplink packets 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 onboard 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 minimum 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 2-1.

Table 2-1: Information Object View of a Spacecraft Command Message File

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

2.2.3.3 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 label file in the PDS' Object Description Language (ODL) format. It can be represented using the information object construct. The information object consists of a set of data objects, such as *SPICE or image files*. Each data object is described by a metadata object, the PDS Label. For the SPICE files, metadata objects include data elements (including their types) 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 file name of the SPICE file must exist in the PDS volume; the orbit number element's value must be a valid orbit number from the mission; and the mission name element's value must be a valid PDS mission. 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 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 2-2.

Table 2-2: Information Object View of a Planetary Data System Product

Data Object		**Metadata Object (Derived from PDS ODL Label File)**		
Name	***Type***	***Data Element***	***Data Element Type***	***Semantic Constraints***
SPICE Files	Set of Ancillary Spacecraft Data Files	File Name	String	Must exist in the PDS 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.

2.2.3.4 Space Link Extension 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 (reference [35]) defines a set of information objects called *service management objects* (shown in figure 2-7) for automating the exchange of SLE-SM information. The service package request includes the *space link session service request*, *space communication service request*, *configuration profiles*, *antenna constraints*, *trajectory prediction reference*, and *retrieval service package request.*

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, and the rest of the objects shown in figure 2-7. Each data object would have a corresponding metadata object. For example, two of the data objects from figure 2-7 are shown in table 2-3, leaving out the rest of the examples for brevity. In table 2-3, the Space Communication Service Request data object has a metadata object associated with it that contains several data elements: *spaceCommunicationServiceProfileRef*, of type URN, that is a pointer to a Space Communication Service profile; *spaceCommServiceStartTime* is the start time for communication service, similarly the stop time, time lead, and time lag, all defined using CCSDS time standard format; *minimumServiceDuration* is the minimum acceptable length of the service session, similarly the preferred duration, also in CCSDS time format as is the *timeReference* data item; and *transferServiceDeferred* and *sequenceOfEventsDeferred* are both of type Boolean. Examples of semantic constraints in the above metadata objects would be to verify that the formats correspond to known mimeType specifications, and to check that the maximum durations does not exceed any prespecified value.

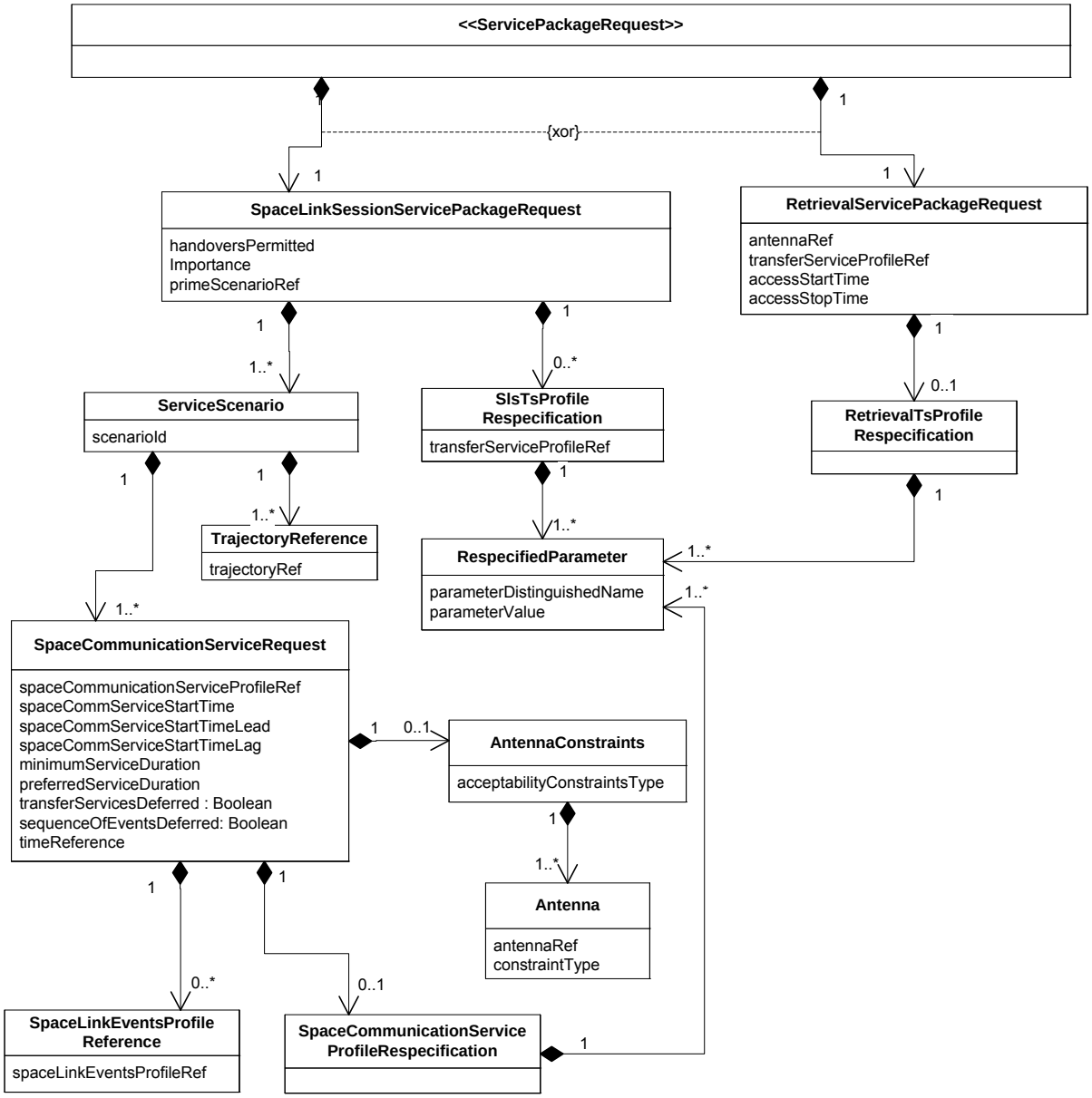


Figure 2-7: Service Package Request Model Overview

Table 2-3: Information Object View of an SLE Service Management Object

Data Object		**Metadata Object**		
Name	***Type***	***Data Element***	***Data Element Type***	***Semantic Constraints***
spaceCommunicationServiceProfileRef	Service Profile ID	Dataset Reference	String256	Must contain pointer to valid profile
spaceCommServiceStartTime	unsigned integer	UTC	CCSDS Time Code	Time must be valid within network and mission visibility constraints.
minimumServiceDuration	positive integer	Duration	CCSDS Time Code	Minimum duration must meet network service constraints
sequenceOfEventsDeferred	Boolean	Switch	Boolean	If SOE profiles are not supported shall be set to NULL
timeReference	enum	Enumerated list	Text	Allowed values are 'absolute' or 'relative'

2.3 MODELING CONCEPTS

2.3.1 GENERAL

Models are important in information architecture because they provide the means to describe and use objects. Without explicit models, objects cannot be examined, understood, or changed accurately, nor can they be compared or integrated with other objects. These capabilities are critical in space data systems because they facilitate the correlative use and exchange of data.

The basic relationship between information architecture models and the objects they describe are illustrated in figure 2-8. The data object is described by the metadata object; both are components of an information object. The metadata object is an instance of a class of objects that are prescribed by (one or more) domain models (e.g., a 'preservation domain model'). The domain model in turn is an instance of a class of objects that are prescribed by a meta-model.

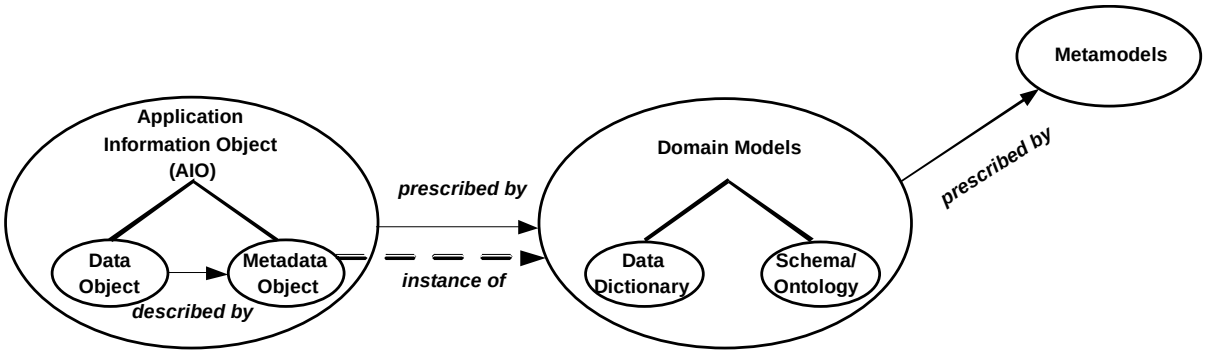


Figure 2-8: Information Object in Context

The hierarchical relationship between objects, models, meta-models, and even meta-meta-models can be simplified using a generalization proposed by the Object Management Group (OMG) (reference [24]). When considering the hierarchy of models illustrated in figure 2-9, any object at level *n* can be described by an instantiation of an object from a class in level *n-1*. For example, the domain model at level M1 can be described by an instance of a unified modeling language (UML) model at level M2.

Models interact within and across levels. For example, ISO/IEC 11179 can be used as a model for a data dictionary. In turn, the data dictionary could be used as a component of a domain model.

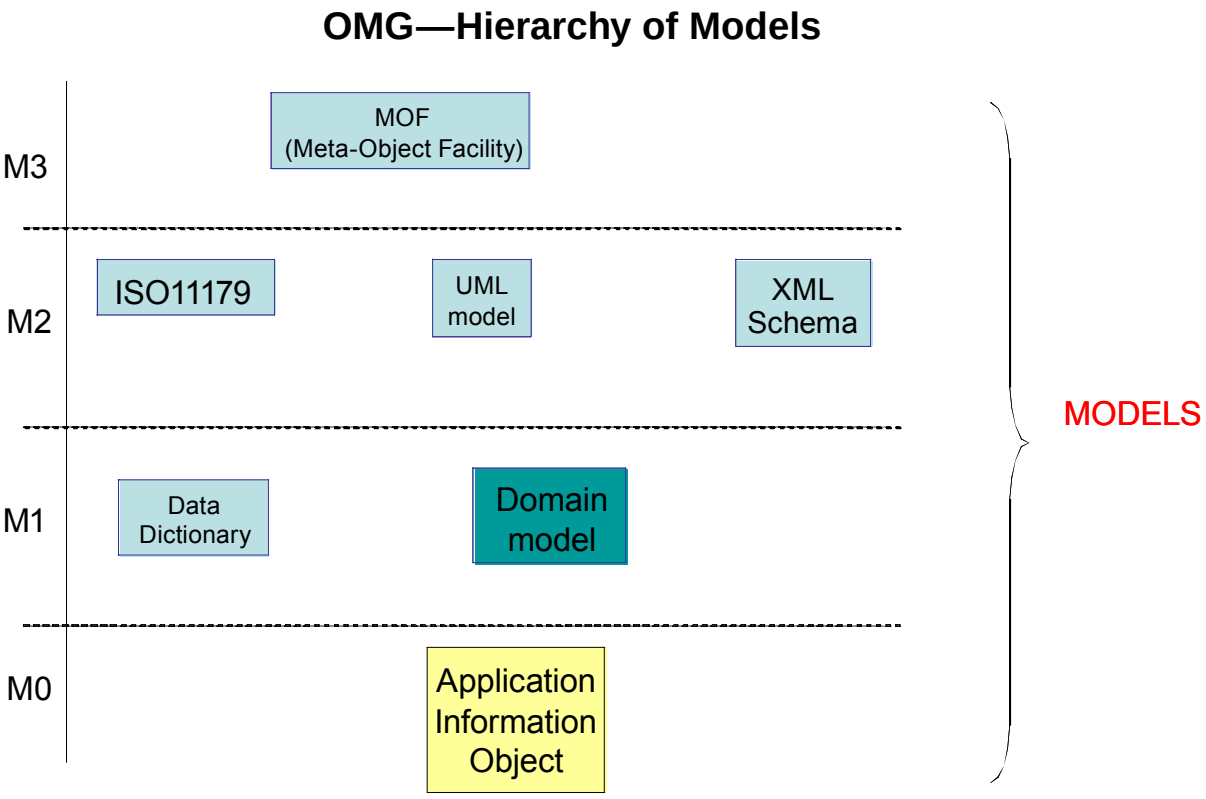


Figure 2-9: Model Hierarchy, Adapted from Reference [24]

In information architecture, the focus is on identifying a set of standard models that meet the requirements for developing information management systems. At the M3 level, meta-object facility (MOF) has been identified as the key model. At the M2 level, UML, the XML meta-model, and ISO/IEC 11179 have been identified as key models for system and domain models, data interchange structures, and metadata registries, respectively.

2.3.2 META-MODELS

2.3.2.1 General

A meta-model is simply a model that prescribes another model. For example, in the generalization illustrated in figure 2-9, UML at level M2 is a meta-model that can be used to develop a domain model at M1. And MOF at level M3 is used to define UML at level M2.

In information architecture, meta-models are important because they prescribe how elements can be compared and examined across domains. If elements did not conform to a particular meta-model, then it would be impossible to guarantee the ability to compare and examine them even within the same domain. Since the ability to compare elements is critical to enabling interoperability of data exchanged between systems, it is necessary that common and/or compatible meta-models be used to describe domain elements both within and across domains. For example, the PDS defines the data element Mission Name described earlier using a meta-model that requires the data element name, a description, and a set of valid values. In order for a query using Mission Name as a constraint to find data in both the PDS and another space science domain, it is critical that the second domain have a compatible meta-model in order to find the equivalent mission constraint that will produce valid results.

In the following subsections, several standard meta-models briefly will be described. These include the ISO/IEC 11179 standard for the specification and standardization of data elements (reference [19]), the CCSDS Data Entity Dictionary Specification Language (DEDSL) (reference [7]), and the XML Formatted Data Unit (XFDU) (reference [25]) model for describing information packages.

2.3.2.2 ISO/IEC 11179

In the realm of meta-models, the ISO/IEC 11179 (reference [19]) standard framework for the specification and standardization of data elements provides a basic foundation for meta-models, metadata registries and how to use them. It specifies general registry functions such as definition, identification, naming, administration, and classification. It provides an accepted base set of attributes needed to describe data elements. As an international standard, it also provides a global basis for data element definition and classification and supports data dictionary interoperability. The specification classifies the basic set of attributes into four categories: *identifying*, *definitional*, *representational*, and *administrative*.

2.3.2.3 Data Entity Dictionary Specification Language

The CCSDS DEDSL provides a specification for the construction and interchange of data entity dictionaries. Its conformance to ISO/IEC 11179 has been documented in (reference [7]) along with an XML specification for its use.

2.3.2.4 XML Formatted Data Unit

The XFDU Structure and Construction Rules is a set of CCSDS recommendations for the packaging of data and metadata, including software, into a single package to facilitate information transfer and archiving. It also provides a detailed specification of core packaging structures and mechanisms that meets current CCSDS agency requirements. (See reference [29].)

2.3.3 DOMAIN MODELS

2.3.3.1 General

A domain model describes objects belonging to a particular area of interest and also defines attributes of those objects, such as *name* and *identifier*. It also defines relationships between objects, e.g., 'instruments *produce* data sets'. Besides describing a domain, domain models also help to facilitate correlative use and exchange of data between domains. Some common space domain models are mentioned briefly below.

2.3.3.2 Planetary Science

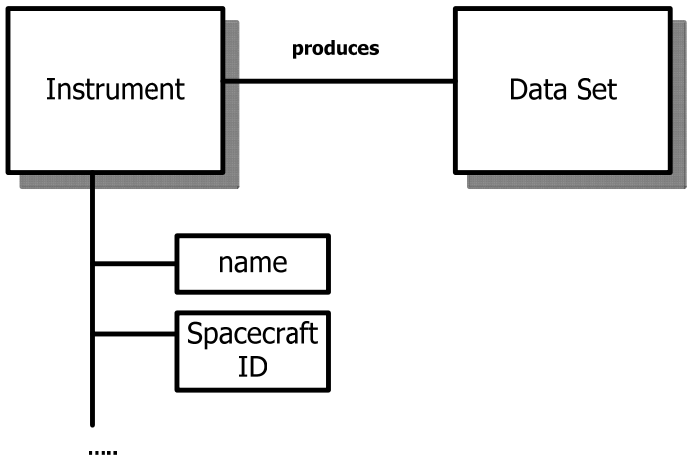


Figure 2-10: Planetary Science Domain Model Example (Simplified)

NASA's planetary science domain model defines objects such as *instruments* and *data sets* and *science users* and their associated relationships (such as instruments produce data sets,

and data sets are distributed to science users). This is illustrated in figure 2-10. The planetary science domain model is defined in PVL/ODL (described in 2.3.4.2).

2.3.3.3 Space Physics Archive Search and Exchange

Space Physics Archive Search and Exchange (SPASE) (reference [27]) is a space and solar physics domain model being developed by an international working group with participation from several national agencies, universities, and industrial affiliates. The SPASE model attempts to define relationships between ancillary data, images, and plots for space and solar physics data products, such as images and data collected about photons and particles.

2.3.3.4 Earth Observing System (EOS) Data and Information System and EOS Core Data Model

The Earth Observing System (EOS) Data and Information System (EOSDIS) (reference [17]) domain model defines data product types, a knowledge base, and a global thesaurus for Earth science terminology to interpret the data products collected in Earth observing systems. Data products include sea surface temperature measurements, global climate measurements, and many other Earth science data products. The EOSDIS Core System (ECS) Core Data Model (reference [14]) was developed as an extension to the earlier EOSDIS domain model in order to specify relationships necessary to handle the sheer data volume (nearly two terabytes a day) that is regularly captured in the EOSDIS system. In the ECS model, data is represented as collections of smaller units, called *granules*. Collections define a series of attributes including, but not limited to, spatial coverage, temporal coverage, and contents.

2.3.4 DATA DESCRIPTION LANGUAGES

2.3.4.1 General

Data description languages are notations used for representing semantic and syntactic data. As such, they provide the necessary implementation level facilities to manipulate and exchange application information objects, and to implement meta-models, domain models and information. Some common examples of data description languages are listed below.

2.3.4.2 Parameter Value Language/Object Description Language

The Parameter Value Language (PVL) (reference [30]) is a CCSDS Recommended Standard for the specification of a standard keyword value type language for naming and expressing information objects. It defines a language that is both human readable and machine readable. This keyword value type language has been used to document domain models in a way conceptually similar to the approach taken by the World Wide Web Consortium's (W3C's) Resource Description Framework (RDF). The Object Description Language (ODL) is a subset of PVL.

2.3.4.3 Enhanced Ada SubseT (EAST)

The Data Description Language EAST Specification (reference [6]) is a CCSDS Recommended Standard that defines a language and syntax for the expression and exchange of information objects, in the form of *data description records* (DDRs). The idea behind a DDR is to provide enough information about data (e.g., its format, size, etc.) to be able to interpret and exchange it in an automated fashion.

2.3.4.4 Extensible Markup Language

Extensible Markup Language (XML) 1.0 (reference [28]) is a W3C specification and syntactic format for data objects formatted in XML, which is a subset, or restricted form of the popular Standard Generalized Markup Language (SGML). XML defines information objects called *entities* which capture data (parsed or unparsed) delimited by XML *tags*, which are named value attributes enclosed by a '<' and '>' symbol respectively. Entities may have sub-entities and *attributes*, which describe related information about a particular entity object, such as its name, or its ID.

2.4 INTEROPERABILITY

The data dictionary plays an important role in making data systems exchange information. A data dictionary for a particular domain provides the data elements for the domain model. In order to support interoperability between domains, it is important to have a common meta-model for data dictionaries so that they can be captured and exchanged in a common way. Further, it is important to recognize that data dictionaries cannot be constructed without a domain model.

The key requirement to enable data-system interoperability is to have common or at least compatible data elements across the respective domain data models. In figure 2-11, two domains and their respective data models are illustrated. The two domains can interoperate, or exchange information, when knowledge exists about data element commonality at the data model level. For example, if both domain data models contain the data element *target name* with 'Mars' as a valid value, then the two domains can exchange information about Mars. The knowledge about data element commonality, depicted as the *interchange model* in the figure, is difficult to acquire and requires domain experts to compare elements from their respective domains for similarities. Often elements will have similar attributes such as *name* and *valid value*s but significantly different interpretations and definitions. These similarities and differences must be understood and documented.

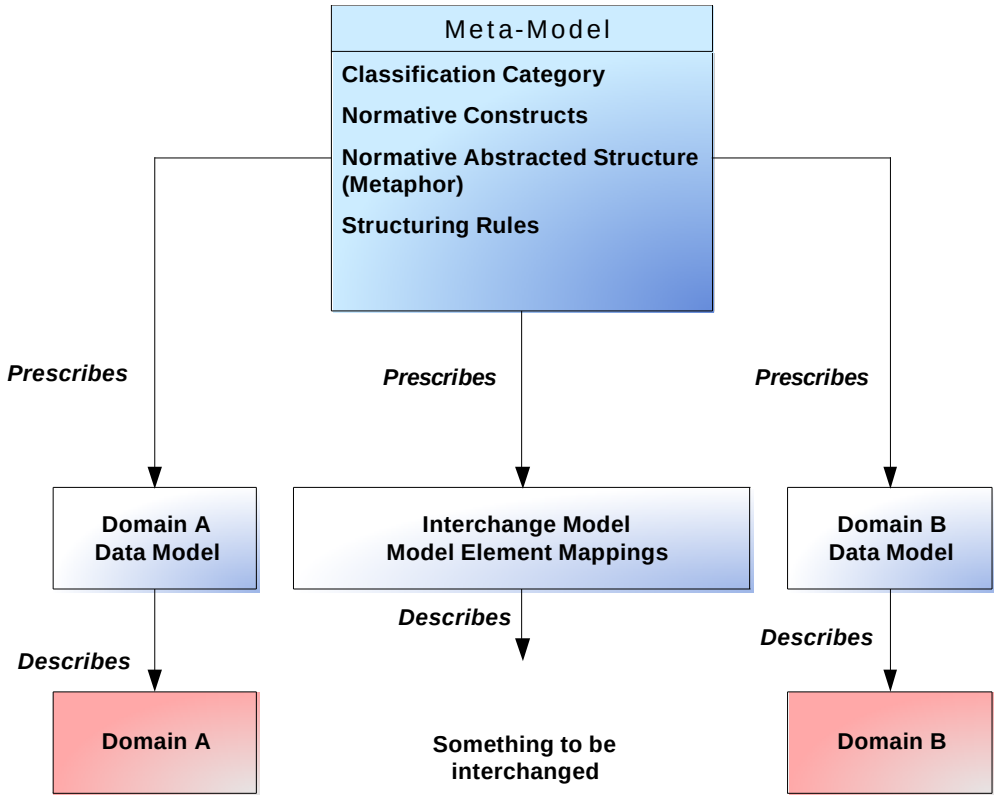


Figure 2-11: Data Models, Meta-Models, and Domains

The process of comparing data elements is made much easier if a single data model (or meta-model) is used to capture the domain data models. It provides a standard notation, syntax, and semantics so that data elements from two different domains can be contrasted and compared. For example, the ISO/IEC 11179 recommendation for the specification of data elements (reference [19]) provides a comprehensive set of attributes for describing data elements and provides a good basis for data dictionary development.

3 SOFTWARE COMPONENTS FOR INFORMATION ARCHITECTURE

3.1 OVERVIEW

This section describes Information Management Objects (IMOs)[3] used for the access, distribution, capture, and management of information objects. Two types of IMOs are identified: *Primitive Information Management Objects (pIMOs)* and *Advanced Information Management Object (aIMOs)*. Generally, aIMOs are constructed from one or more pIMO components. pIMOs are active objects capable of *putting, getting,* and *finding* information from the underlying data stores. aIMOs are complex objects, composed from one or more pIMOs, that enable basic capabilities of information architecture, including *ingestion*, *retrieval*, *processing*, *distribution,* and *querying* of data objects, metadata objects, and information objects. Although this set of capabilities is not the entire set of capabilities that could be derived, it is meant to be a framework of building blocks from which further complex capabilities are defined. More specifically, this section will describe the functional components used to construct space information systems. The discussion of the actual architectural topologies and orchestrated use of real systems constructed using these components is left for a later reference standard.

3.2 PRIMITIVE INFORMATION MANAGEMENT OBJECTS

3.2.1 GENERAL

pIMOs are simple functional components capable of manipulating their underlying data storage using *put*, *get*, and *find* operations. There are two types of pIMOs: *data store objects (DSOs)*, and *query objects (QOs).* These objects (components) are called *primitive* since this document does not explicitly identify any of their architecturally relevant sub-components. Both of these pIMOs operate on a *physical data storage* component.

A physical data storage component is a hardware or software component responsible for storing data. Devices such as tape drives, hard disks, solid-state recorders, RAM, flash memory, and the like are all examples of physical data storage components. There are two basic parts of physical data storage components:

- **Memory**—the physical location of the data in the data storage (labeled as ‘D’ in figure 3-1);
- **Local Identifiers**—the index catalog of pointers to memory containing data objects (labeled as ‘H’ in figure 3-1).

These parts enable low-level access to physical data storage to 1) place data objects into memory locations, and 2) index those locations for use in search and retrieval processes. The organization of a physical data storage component is shown in figure 3-1.

[3] The words ‘objects’ and ‘components’ are used interchangeably in this context.

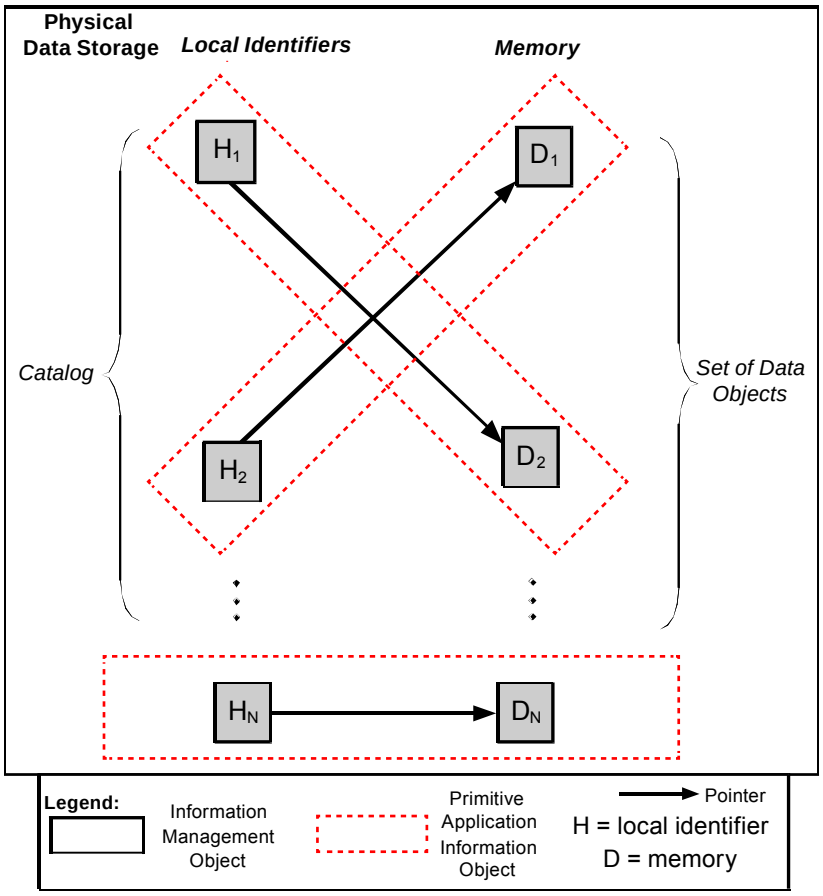


Figure 3-1: The Internal Structure of a Physical Data Storage

While it may not be entirely obvious in devices such as solid-state recorders, the evidence of low-level catalogs that take the form of local identifiers is clear. Without some sort of address with which to locate information stored in memory, the information would never be found.

3.2.2 DATA STORE OBJECT

The *data store object* (shown in figure 3-2) is attached to a physical data storage and supports *putting* and *getting* information. Figures 3-3 and 3-4 depict the *put* and *get* operations of the DSO, respectively. The *get* operation takes a *local identifier* as input (ranging from a simple memory address to a string identifier) and returns the data object (shown as 'DO' in figures) residing in the addressed memory location as an output. The *put* operation takes a data object as input and, upon completion, places the data object in a free memory location (labeled as 'local identifier' in figure 3-3) determined by the catalog and ingestion process of the underlying physical data storage. The local identifier is then returned back to the caller.

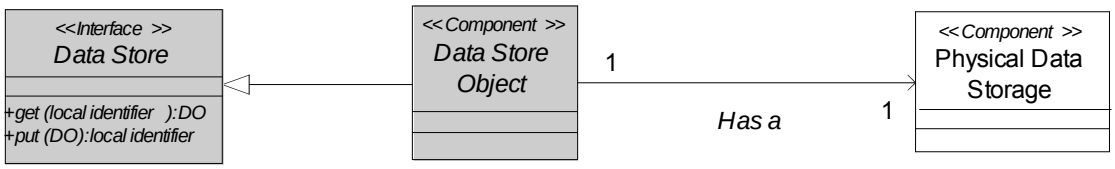


Figure 3-2: A Data Store Object

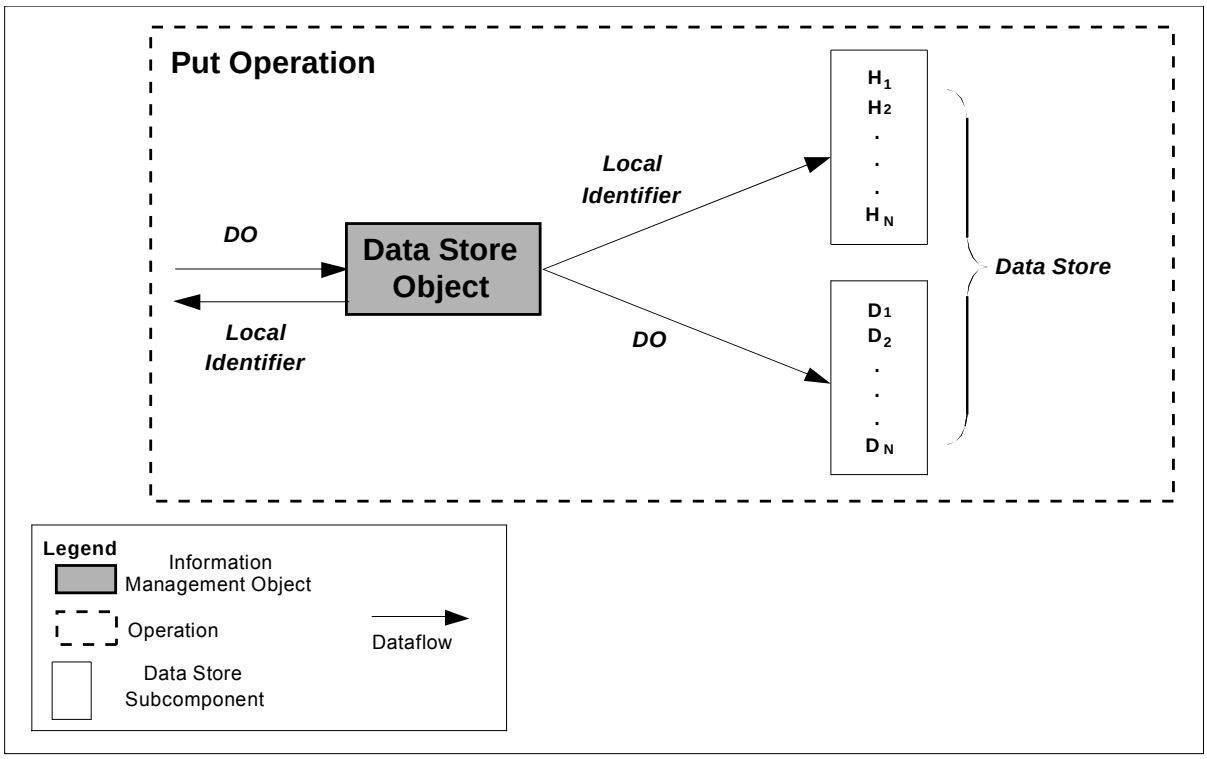


Figure 3-3: The *Put* Operation of the Data Store Object

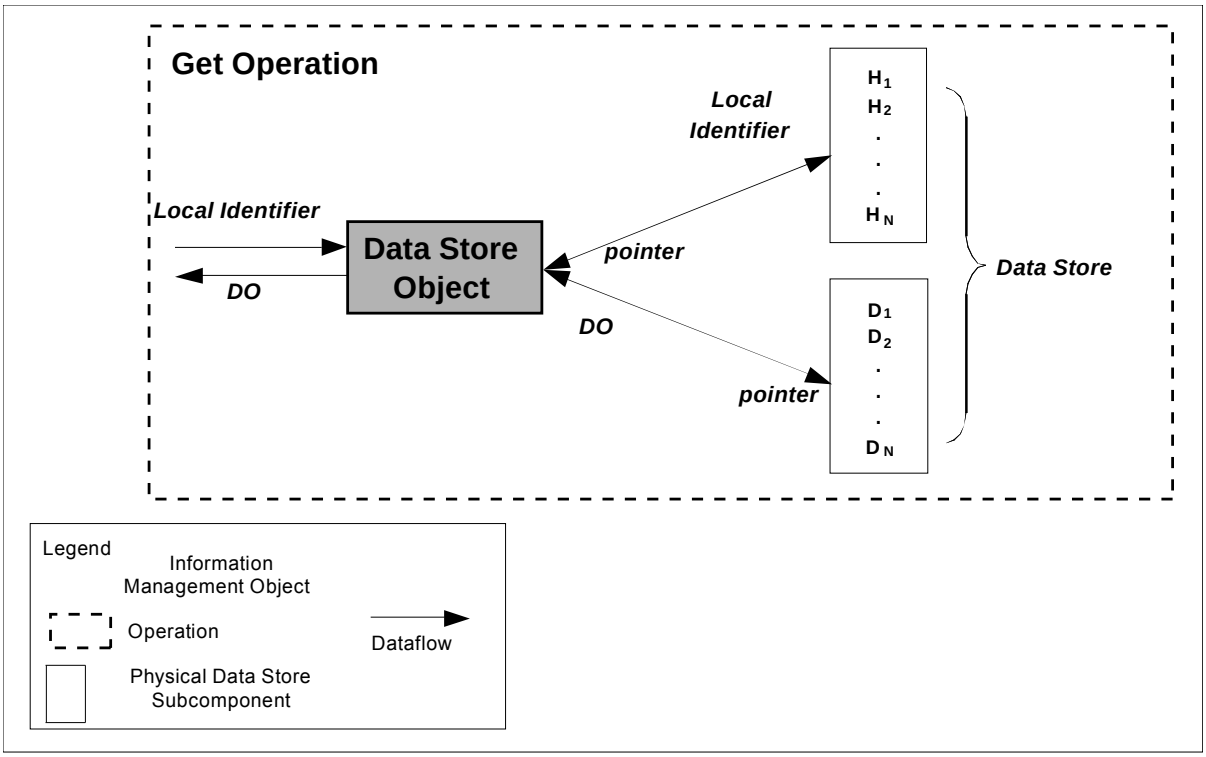


Figure 3-4: The *Get* Operation of the Data Store Object

3.2.3 QUERY OBJECT

The *query object* shown in figure 3-5 enables retrieval of data objects. Data objects (shown as DOs in figure 3-5) are retrieved using the *find* operation. The find operation takes an *expression* parameter representing a specific search criterion for the underlying physical data storage. Each matching data object is then returned to the caller of the find operation. A find invocation may return zero or more data objects. Figure 3-6 visually describes an example of the find operation and the data flow between the query object component and the respective physical data stores it communicates with.

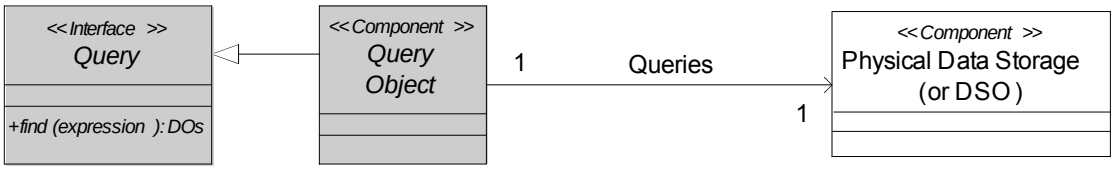


Figure 3-5: A Query Object

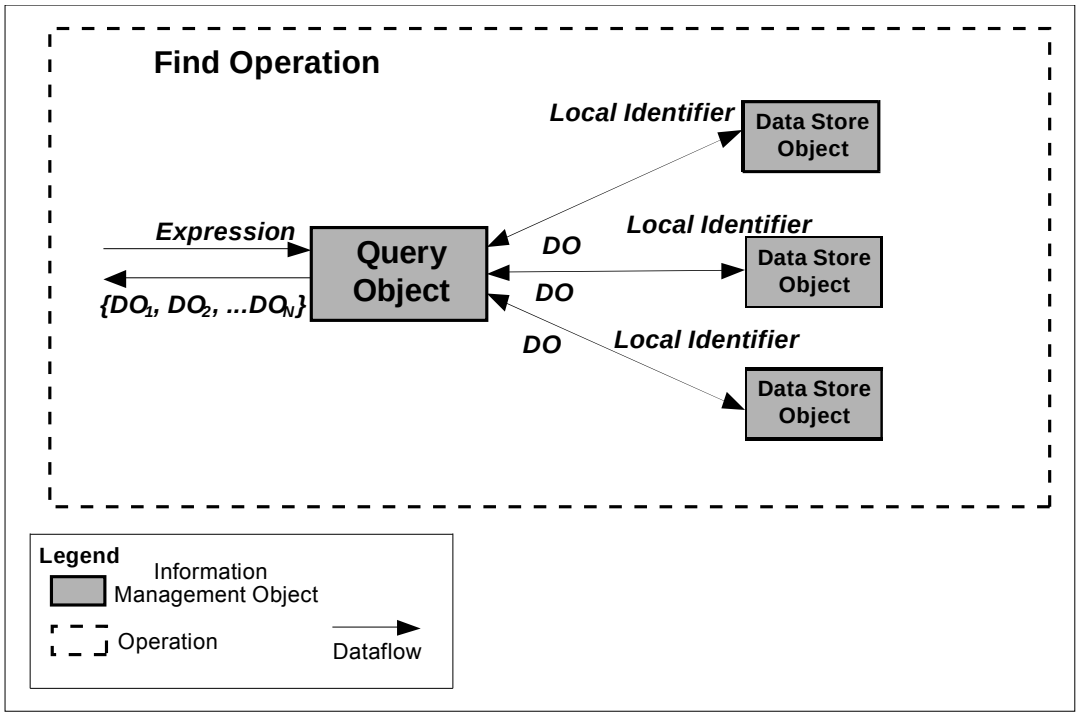


Figure 3-6: The *Find* Operation of the Query Object

3.3 ADVANCED INFORMATION MANAGEMENT OBJECTS

3.3.1 GENERAL

aIMOs are components composed from one or more pIMOs. aIMOs leverage pIMOs' primitive data store and retrieval functions to produce 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 aIMOs presented in this document is not meant to be comprehensive. There are other aIMOs, but the set presented here represents a sound cross-section of advanced components that span the typical usage scenarios involved in data systems. In the rest of this section, the following aIMO components are discussed in more detail: *Repository Service Objects*, *Registry Service Objects*, *Product Service Objects*, *Archive Service Objects*, and *Query Service Objects.*

3.3.2 REPOSITORY SERVICE OBJECT

3.3.2.1 General

The *repository service object* component is depicted in figure 3-7. 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 several properties that are typically considered *nonfunctional*. These properties include *scalability*, *dependability*, *uniformity*, and other quality attributes. In this context, repository service objects provide the

same *get* and *put* methods that the data store object provides. However, whereas a data store object may not scale across many underlying physical data stores, may not be dependable 24×7, and may not provide a uniform software interface, a repository service object is responsible for delivering necessary quality of service in each of these nonfunctional properties.

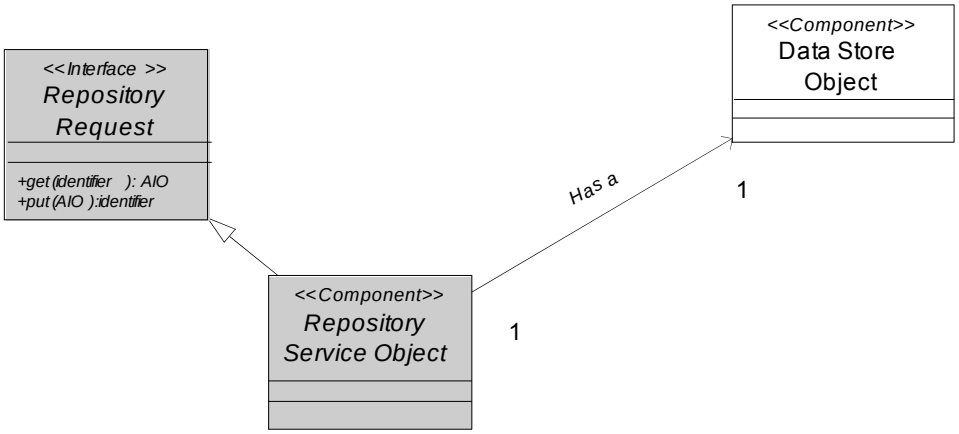


Figure 3-7: Repository Service Object

A repository service object's primary interface is a *repository request* that can be used to manage information objects (IOs). 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.

3.3.2.2 A Taxonomy of Repository Service Objects

Information architecture makes a distinction among different types of repository service objects along several dimensions. There are three main dimensions in a repository service object taxonomy, *repository object type*, *object properties*, and *object description*:

a) Repository service objects are identified via their type. A *repository object type* provides a quantifiable grouping for a family of repositories with similar functional and nonfunctional properties. This document identifies three key repository types: *data store*, *operational archive*, and *long-term archive*.

b) The *object properties* dimension serves as a general grouping of various functional and nonfunctional properties a repository might have. This object properties dimension covers the entire scope of properties for these repositories; however, other properties will be categorized for comparison and classification of other different repository service objects. Potential dimensions of repositories include the following:

   1) *compositionality*, referring to the lower-level and higher-level organization of the sub-components of a repository;

2) *supported data objects*, referring to the type of data objects that a repository is responsible for storing;

3) *permanence*, referring to the nonfunctional property of how long the data is guaranteed safe and reliable shelter within a repository; and

4) *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.

c) *Object description* identifies key services and responsibilities of the repository when deployed together with a set of other software components. Table 3-1 lists the current taxonomy and classification of repositories.

Table 3-1: A Taxonomy of Repository Service Objects

Repository Object Type	Object Properties	Object Description
Data Store	Primitive Component (e.g., Database Management System [DBMS], Solid-state Recorder and/or File system).	Basic data store component described in 3.2 sits behind data store object and supports repository interface to *get* and *put* 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., typically less than 1 year), and allows retrieval of products.	Advanced component supporting retrieval of possibly complex data products, including their metadata. Repository where both writes 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).

3.3.3 REGISTRY SERVICE OBJECT

3.3.3.1 General

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* described in section 2. 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 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 that satisfy a particular query expression provided by the user of the *metadata Query* interface. Figure 3-8 depicts a registry service object.

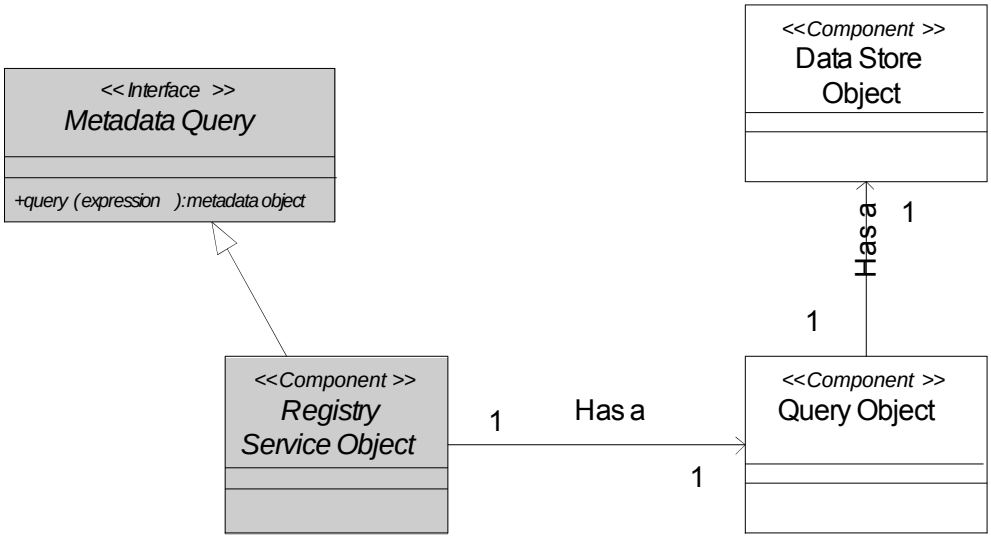


Figure 3-8: A Registry Service Object

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

3.3.3.2 A Taxonomy of Registry Service Objects

This taxonomy identifies three main classes of registries and then classifies 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 (recall the description of meta-model in 2.3.2). 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 data elements. Constraints and values are specified either implicitly by querying the data element properties, or explicitly by specifying the data element's ID (see reference [19]).

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 technical standards such as WSDL (reference [28]) 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 [11] and [22]. 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 (reference [3]) 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 nonfunctional classification unspecified at this point. This type of classification of nonfunctional registry service properties is very important. The taxonomy of registry service objects is summarized in table 3-2.

Table 3-2: A Taxonomy of Registry Service Objects

Registry Type	Return Object Types	Query Interface Parameters
Metadata Registry	Data dictionaries, data elements, (meta-) metadata objects	Query for data element properties, 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

3.3.4 PRODUCT SERVICE OBJECT

The next aIMO is the *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 for 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 3-9) 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 3-9.

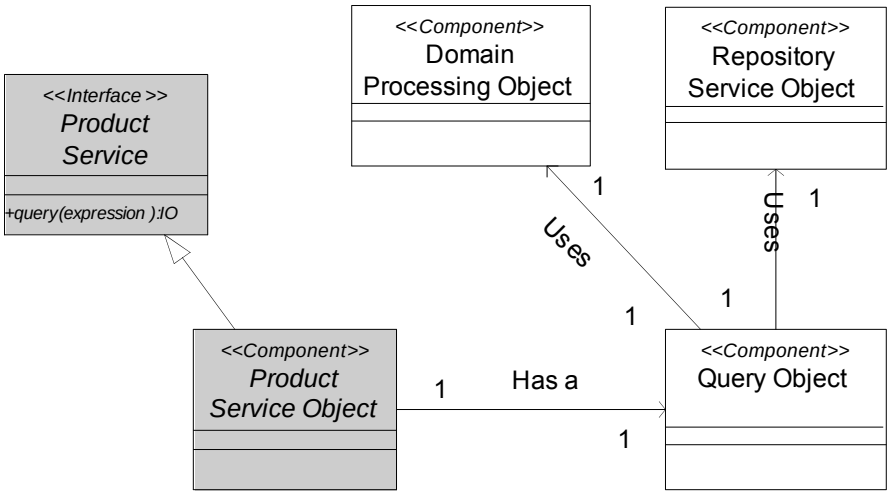


Figure 3-9: A Product Service Object

3.3.5 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 3-10). 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* (see references [4], [12], and [13]) and can execute externalized objects such as the *domain processing object* discussed in 3.3.4. 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 archive. In fact, an instance of this type of 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 3-10.

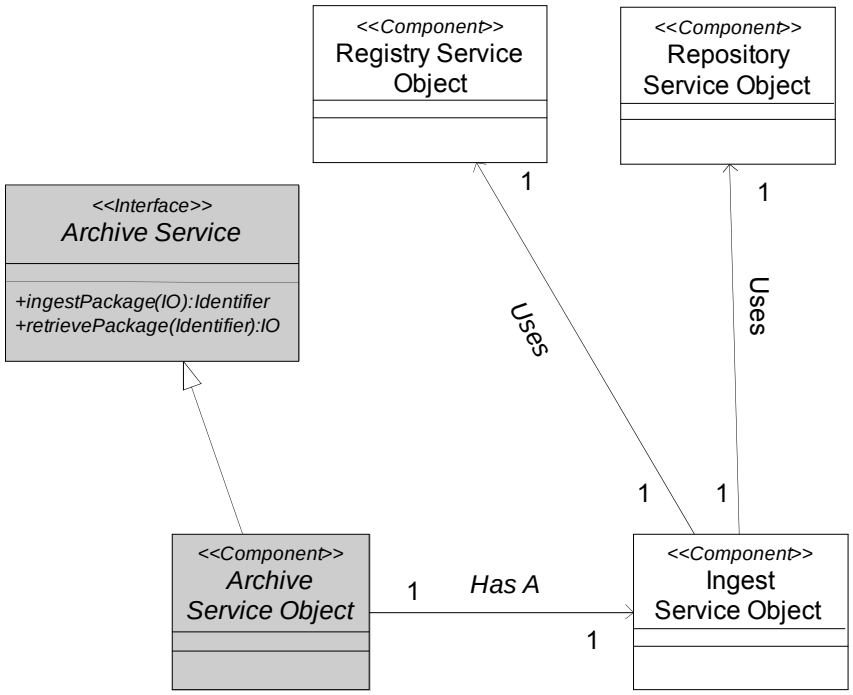


Figure 3-10: An Archive Service Object

3.3.6 QUERY SERVICE OBJECT

The final aIMO defined in this document is the *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 3-11) 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 3-11 and are discussed in 3.3.4 as an externalized component of the product service. Figure 3-11 depicts a query service object.

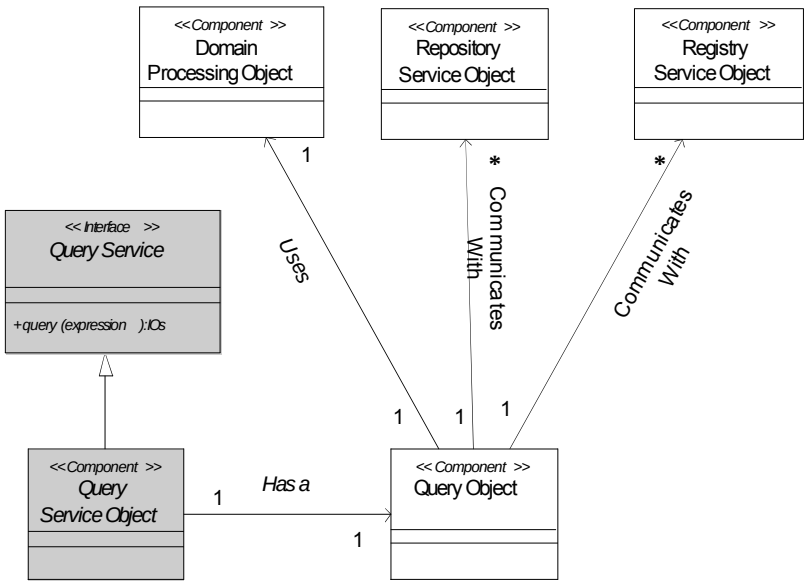


Figure 3-11: A Query Service Object

4 DISTRIBUTED INFORMATION SERVICE ARCHITECTURES

The movement towards distributed services architecture has several implications for CCSDS. Primary benefits include the ability to develop separate systems while improving interoperability between components of a system through loose coupling. Service-Oriented Architectures (SOAs) have become a regularly used term to describe such systems, but a critical point is that 'SOA' generally describes an architectural style for composing such systems and not an interoperable framework, per se. Defining an interoperable SOA framework is a natural progression of systems towards interoperability of applications and systems across a distributed network.

An information service architecture promotes several goals for the composition of information systems based on CCSDS standards, including:

- constructing systems using well-defined standard interfaces;
- enabling agency autonomy in their implementations;
- supporting federation of space data system services across agencies;
- supporting service discovery between organizations and agencies;
- enabling loose coupling of services and their implementations;
- supporting reusability and development of common agency product lines.

At the same time, development of such system requires coordination on several fronts, including:

- shared messaging infrastructures;
- common data definitions of the contents of messages;
- common services behaviors;
- common service interfaces and methods;
- information services to support service registration and discovery;
- explicit methods for secure access of services and exchange of information.

Within CCSDS, the CCSDS Information Service Architecture is defined as *an architectural pattern that decouples common information services, models and middleware from applications to improve reuse and interoperability between applications*. It prescribes:

- common information services and components (described in section 3);
- common meta models (as described in section 2);
- common information models (as described in section 2);
- distributed messaging middleware (described in this section).

4.1 INFORMATION SERVICE ARCHITECTURE VIEWS

4.1.1 OVERVIEW

The Information Service Architecture can be best defined with three key specific views including *information*, *function*, and *service*. The use of views to describe architectures is well defined by ISO/IEC/IEEE 42010:2011 (reference [32]) and by RASDS (reference [34]).

4.1.2 INFORMATION VIEW

Within RASDS, the *information view* recommends that any information system explicitly define its information architecture using a well-specified information model. This includes definition of the *information objects* as defined in section 2. The information objects are based on common and domain-specific models that govern definition of the semantic and syntax of their content. Well-designed systems that exchange information across distributed services need to ensure that their interfaces and the content that is exchanged across these interfaces is governed by explicit information models.

4.1.3 FUNCTIONAL VIEW

The *functional view*, as it relates to information architecture, provides a set of standard information management components that allow for construction of applications and services within distributed systems. The term *components* is used to mean any modular element of a software system, but in particular, from an information services perspective, they refer to components that can be used to implement common information management functional objects as described in section 3.

4.1.4 SERVICE VIEW

The *service view*, as it relates to information architecture, provides a set of software-based services, constructed from software components, that are deployed within a distributed system, carry out a set of relevant functions, and expose services with a well-defined interface. Services include both generic information infrastructure services such as registries, repositories, and messaging services, as well as domain-specific services. Within distributed systems, services with standard interfaces are used to support flexible deployment of software systems that support the access and exchange of information as well as computation. For data and information-oriented systems, standard components that capture, register, discover, and exchange information are critical.

Layering domain-specific application services on top of an information service architecture permits migration of domain services into a distributed, information-driven architecture. As systems become more of a distributed architecture, it is critical that application-layer components be built that leverage explicit distributed computing principles to support their deployment as distributed services within an agency or multi-agency model. Figure 4-1

below, shows the relationships of services, based on the OASIS Reference Model for SOA v1.0 (reference [33]) definition.

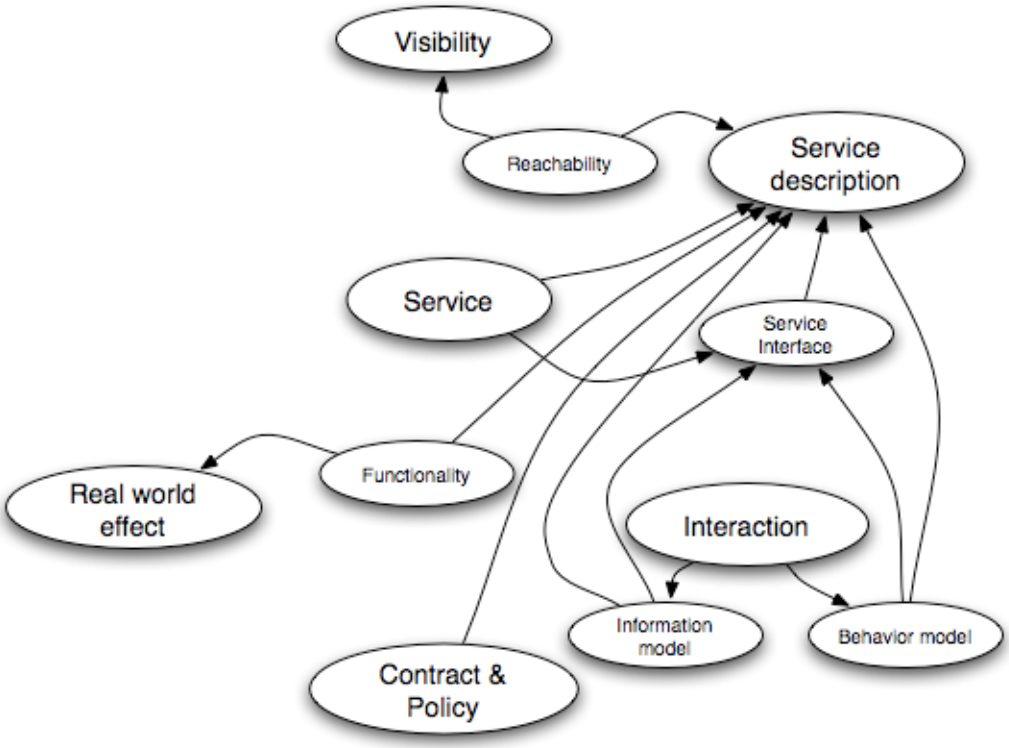


Figure 4-1: Service Description Conceptual Model

4.2 SERVICES AND PATTERNS

4.2.1 GENERAL

A service-oriented architecture encapsulates services so that they expose messaging interfaces that allow for their functions to be accessed and executed over the network. Figure 4-2 provides logical/static and runtime views of a service-oriented architecture that shows a set of components that are used to enable interoperability between distributed applications through a messaging middleware.

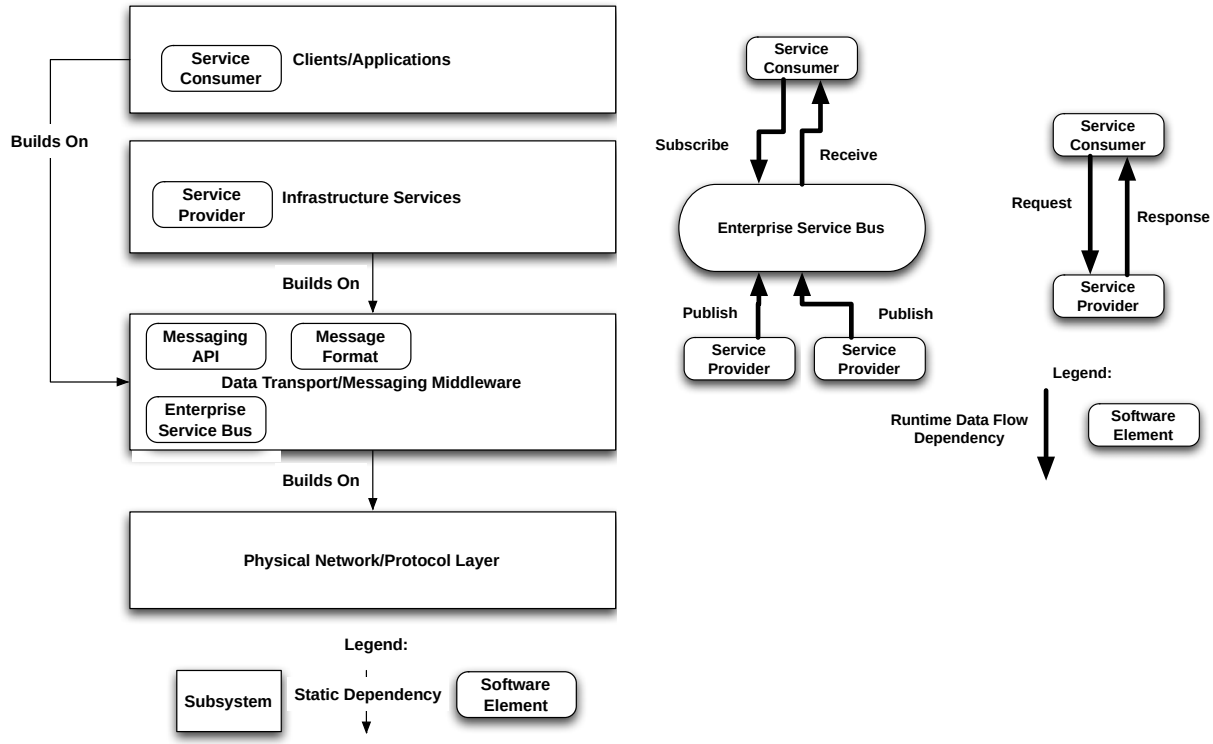


Figure 4-2: Static View (Left) and Runtime View (Right) of an Information Service Architecture

Popular architectural patterns in an information service architecture involve publish-subscribe, request-response, as well as more specific patterns such as Client/Server or Model-View Controller (MVC). These patterns have been implemented in various ways, emphasized by separating the service layer from the application layer. They employ services that are decoupled from applications allowing for access to computational and information services that can be provisioned both within and between enterprises. Critical to accessing these services is defining standards that allow for their use in a common manner which encompass the interfaces and the protocols on which they are built.

The services aspects define a number of common services supporting various patterns that are crucial to implementing a distributed information system. While space-domain specific services are critical to supporting the business of the capture and exchange of *space information objects*, common information services are necessary to enable such exchanges cross enterprises. These services include:

– registries;

– repositories;

– messaging middleware;

– orchestration and workflow.

4.2.2 REGISTRATION OF SERVICES AND INFORMATION OBJECTS

Registries provide for the registration of information objects and services. This includes registration of models, data or services that enable discovery, information sharing, and computation. Within the context of an information services architecture, a service registry is critical to registering the existence and access points for services such that client applications can discover and bind to such services. Several standards exist that allow for registration of services, including the Universal Description Discovery and Integration (UDDI) and the Electronic Business using eXtensible Markup Language (ebXML). These standards prescribe mechanisms that allow for the annotation and registration of services such that they can be discovered both by humans and remote applications. One of the critical advantages is providing specific logical-to-physical mappings whereby applications use logical identifiers to request access to services and are returned a physical entry point, which is necessary for binding to that service.

In addition to service registries, other registries are important to provide for registration of both models and their associated attributes as well as actual information objects. Model and metadata registries provide registration of the components of a model including data elements, relationships, schemas, and the like. This is important in enterprise-scale applications because common models are necessary to support exchange of information in a consistent manner. Such exchanges, whether messages or content, should be derived from models that support both the messaging and domain in which the application is running. Within space data systems, messages should be built on top of standard messaging models and the content should be specific to space data systems, built using common CCSDS information models that support such areas as service management, navigation, mission operations, etc. Registration of the XML schema and associated elements is important to ensure applications are sharing, using, and building schemas that are captured and versioned in common registries.

At the information object level, information objects should also be registered in registries that provide common interfaces within distributed environments that support access and sharing as needed to meet the business cases. Examples within space data systems include capture and sharing of spacecraft identifiers and other standard values. More broadly, within mission and science operations, capture and sharing of engineering and science data can also be performed using common registries such that the data can be shared and accessed both within an enterprise and by federated partners such as other space agencies.

4.2.3 REPOSITORIES SERVICES FOR INFORMATION OBJECTS

Registries and *repositories* are highly related (see 3.3.2 and 3.3.3). While registries provide mechanisms for discovery and access, repositories provide mechanisms for the physical storage of information objects. While multiple patterns exist for the use of registries and repositories, a critical pattern is the registration of metadata portion of an information object with the data object being stored in a repository.

4.2.4 MESSAGING SERVICES

Messaging services, within an information services architecture, are key to decoupling the application from the service layer. Messaging services carry requests between clients and servers that both support the messaging interface. As mentioned, both request response and publish/subscribe models are applicable to an information service architecture. Request/response patterns are largely built around standard messaging protocols whereby applications subscribe to events that are notified when those events occur. In this case, applications and other clients of the services run asynchronously with a thread of execution that monitors a message queue and acts on the message when it arrives. This pattern is often referred to as an *enterprise service bus (ESB)* because applications and clients plug into a software bus and are notified via standard messages. The Java Messaging Service (JMS) is a popular messaging specification that is supported by several vendor and open-source efforts for implementing a messaging bus that exposes a standard API. This API focus of the specification provides interoperability only within the one product and not across different vendor instances of the specification. Enterprise buses are often tightly coupled and employ orchestration mechanisms, which are discussed later. This can be advantageous to a tightly integrated environment, but less advantageous in scenarios where multiple enterprises need to be loosely coupled. Figure 4-3 shows the concept of an enterprise bus.

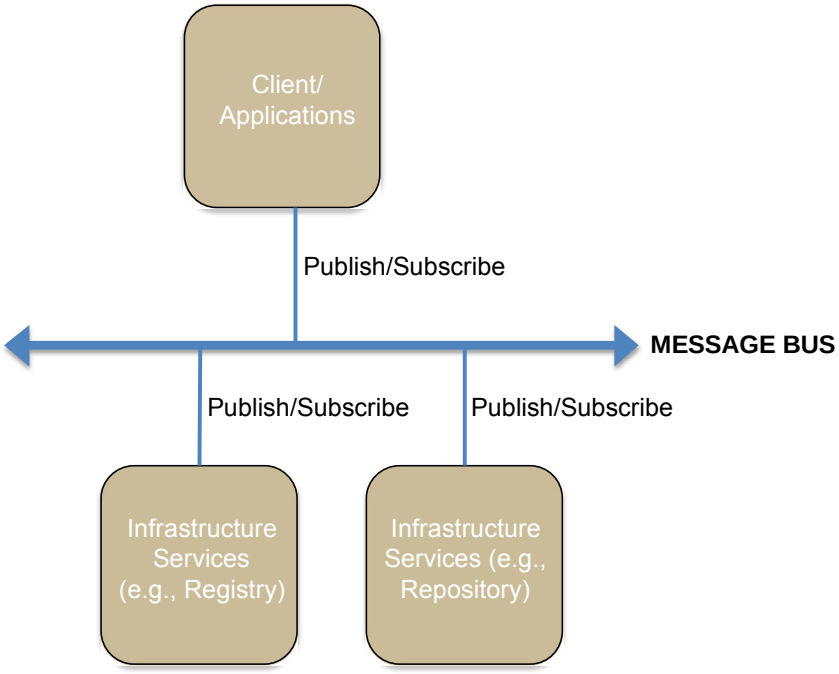


Figure 4-3: Information Services Connected to a Software Bus

In contrast to the ESB publish-subscribe model is the RESTful request-response model, where RESTful refers to a model that conforms to a specific usage of the HTTP protocol. This model uses an interoperable protocol specification and is organized for loosely coupled environments, which significantly contributes to the great scalability of systems like the

Web. This pattern is generally synchronous in that a client sends a request, is blocked, and then a result is returned. However, the RESTful request-response model is increasingly incorporating client-side non-blocking mechanisms and is also becoming more prevalent in sharing data between enterprises. The science community, for example, is largely employing RESTful approaches for sharing scientific data across scientific data systems. This model also has less orchestration in that in many cases of the modern World-Wide Web (WWW), state is not maintained between subsequent calls. Web Service (WS) standards including SOAP and REST have been built on top of the standard Hypertext Transfer Protocol (HTTP) of the WWW. Each of these employs a request-response pattern. The SOAP-related standards, known as the WS stack, provide quite a bit of sophistication in terms of supporting a variety of standards from security to messaging. The REST-related specification is generally viewed as an extension of HTTP with simple messaging intertwined in the request and response as shown in figure 4-4. Both are deployed and both have their benefits and uses.

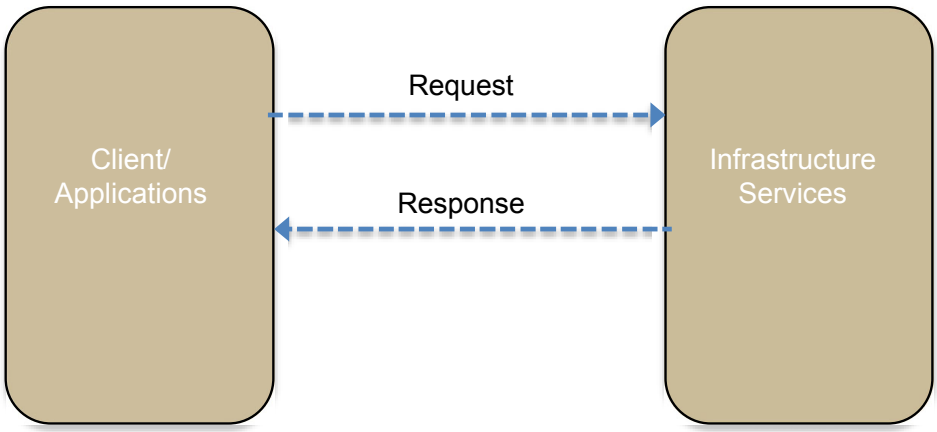


Figure 4-4: REST-Based Distributed Architecture with Information Services on the Back-End

In both the above messaging patterns (RESTful request-response, ESB publish-subscribe), the message protocols are critical to supporting the enterprise. At the protocol level, there are standard specifications, but is run on top of those specifications, in terms of the type of messages and their content, also needs to be defined and specific to the domain. As such, it is important to both define the architecture as well as identify the messaging patterns, protocol and content that is needed to support space-domain related functions.

4.2.5 ORCHESTRATION AND WORKFLOW

Orchestration allows for a set of functions to be executed in a distributed architecture following a set of rules that are instantiated as part of the messaging system. In a publish-subscribe pattern, orchestration allows for events to be triggered based on a specific workflow such that services are invoked and accessed in a coordinated manner. This is important for developing domain-specific workflows that direct the execution of services. The use and deployment of a service architecture benefits from employing orchestration as

engineering and science data move through an end-to-end space data-system pipeline executing generation and capture of command, engineering, and science data products. The processing and generation of data products is increasingly employing complex workflows that manage a set of distributed compute resources, allowing these resources to be allocated and used in conjunction with specific workflows through a distributed service-oriented architecture.

4.3 INTEGRATED INFORMATION SERVICE ARCHITECTURE

Figure 4-5 shows a layered approach to communication among mission operations application-layer, domain-specific services and to underlying infrastructure services. Each of the mission-oriented application functions can be instantiated with a common set of information services coupled with information models for their local deployment. These models dictate the structure, organization, and content of the information objects that flow between the application functions. What is not dictated is how this is distributed. The decoupling between the mission operations application functions and the underlying services is critical to supporting an integrated enterprise that involves distributed operations. The reuse of common information infrastructure services is useful to quickly building and deploying information systems. The integration of a messaging layer brings this together; however, the architecture does not prescribe one specific messaging infrastructure, but rather supports different messaging implementations and patterns, as just discussed in 4.2.4.

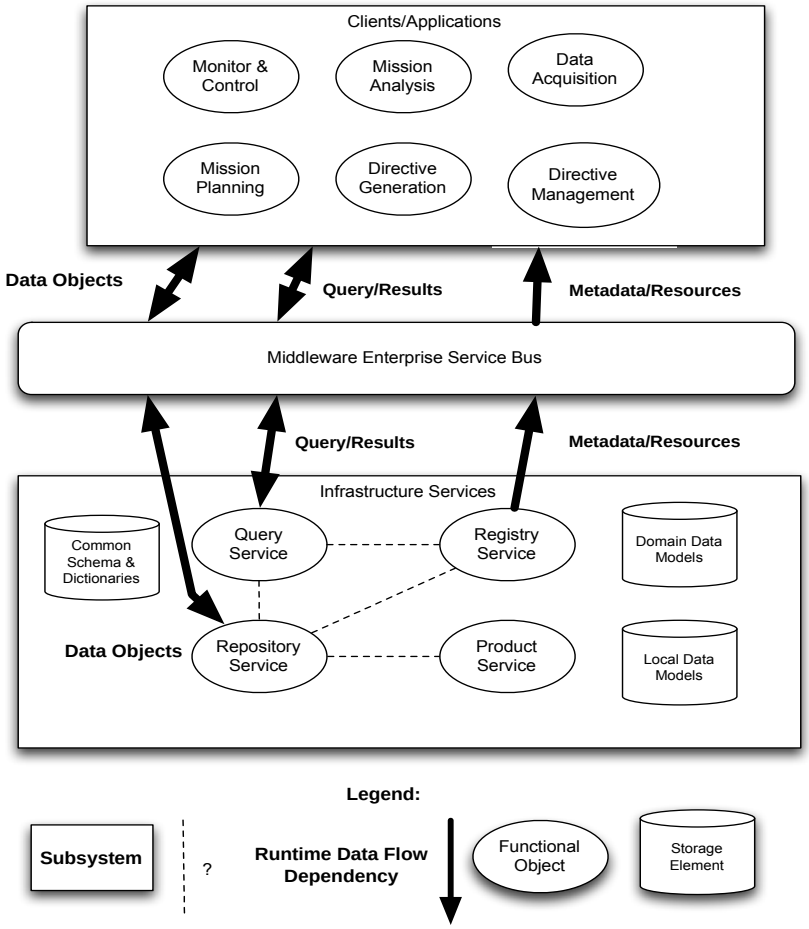


Figure 4-5: Space Domain Functions and Information Services in a Distributed Environment

Figure 4-6 below shows a layered view of the CCSDS-related services that abstracts out the messaging middleware and the information infrastructure. This is used by the application services and the applications themselves.

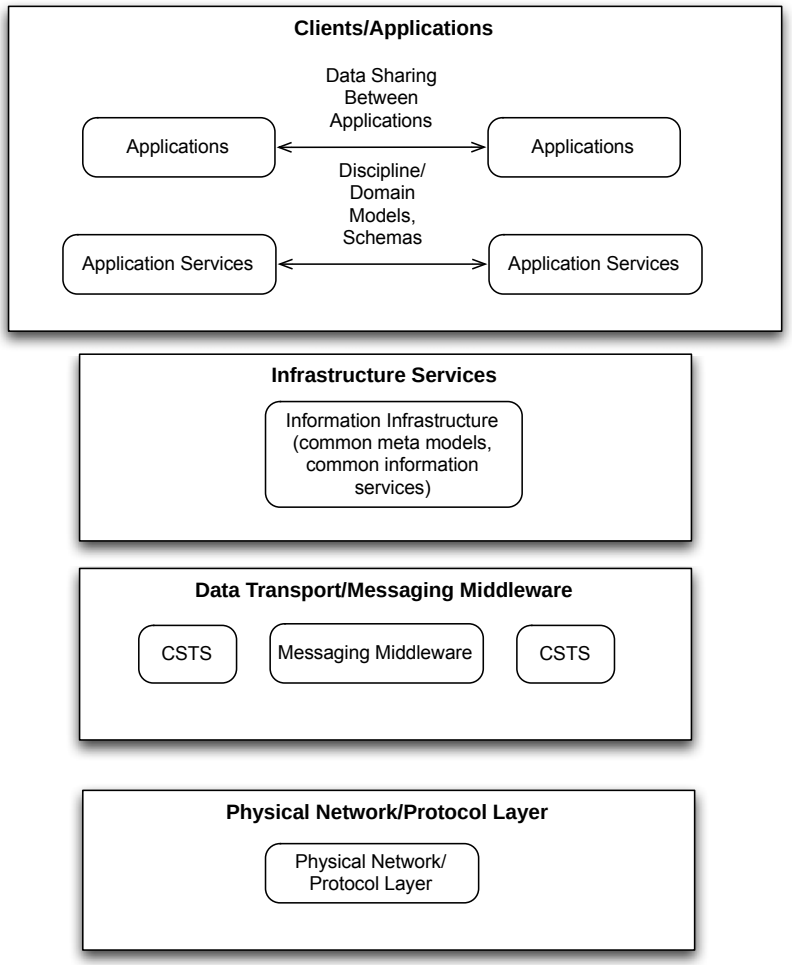


Figure 4-6: Information Service View of the CCSDS Layered Stack

Identification of the separate layers in the architecture is intended to ensure separation of critical elements to support deployment and movement towards an extensible and interoperable CCSDS information service architecture. This includes identifying the following:

- standards related to the messaging middleware so applications, infrastructure, and services can be decoupled;
- standards related to the definition of application services;
- standard information infrastructure components that form the basis for application services;
- common models and meta-models that support the exchange of information between application services.

4.4 INFORMATION MODEL-DRIVEN ARCHITECTURES

Clearly identifying and standardizing the information architecture aspects of a service architecture is critical to moving to data-driven approaches. This is particularly important in a distributed information system where information objects are exchanged across application environments through well-described interfaces, as has been defined in section 2. In data-driven approaches, services and processing are configured by the information model itself. As a service architecture is scaled across an enterprise, the need for moving towards data-driven approaches increases. It is critical that the content being exchanged across an end-to-end space system is not embedded in the core software, but rather described at the content layer. As such, this information service architecture prescribes exchange of information objects on top of services.

The process of information modeling involves applying both specific domain knowledge and information modeling theory to create models that meet the specified requirements. For example, a data model for a magnetometer time-series data product must prescribe the information necessary to make the data scientifically useful. This information might include instrument knowledge and characteristics as well as data structure, provenance, and the location and context within which the data was collected.

Examples of data modeling theory include the Entity-Relationship (E-R) Model, the UML, and ontologies based on the Web Ontology Language (OWL). Ontologies are particularly useful for modeling complex scientific domains and allow for the capture and organization of semantically rich information models in a form that can be reasoned about and transformed into other forms, such as XML schema, for direct use in a deployed system.

To implement a data-driven information service architecture, each data item must be modeled. Subsequently, the model must be transformed and used within the system to collect and validate the prescribed information. The information model must also be able to change in pace with the evolution of the scientific research and technological methods of the domain. The technology used to implement the information system will often change at a different speed, frequently faster, than the domain. This requires that the information model must remain independent of the technology, languages, or notations into which it is implemented or expressed.

To support system longevity, the information model must be scalable to handle increases in size, sophistication, and complexity of the underlying domain. It must also be responsive by maintaining the upper bound on the semantic richness needed for capturing domain knowledge. It must also allow the export of its information to diverse notations and languages for use by both machines and humans for processing and documentation, respectively.

Two common requirements for data systems are the need for data system interoperability and data correlation. Shared data standards are fundamental to meeting these requirements. The information object is a model that prescribes the metadata needed to describe a data object—an instance of actual data such as the time series above—so that it is useful both for current and future users.

The information service architecture defines the domain model to include a schema and attributes that are specific to the domain (e.g., navigation, service management, etc.). The current practice is generally that each domain maintains and publishes its own schema. There is no regard, however, to how these schemes fit together, and they are often maintained as a pseudo information model within the specification language itself, such as XML schema. What is needed is to derive those schemas from a set of information models that are maintained in a modeling environment, decoupled from any specific implementation language such as XML. Derivation of information objects from the domain model is critical to supporting interoperability. While the services provide the functional capabilities necessary to discover, access, and exchange information, the content of the information that is exchanged needs to be derived from the model. Furthermore, how the model is organized can be further defined by a standard governing meta-model.

Figure 4-7, which is described in detail in section 2, dictates that a domain model, which may be governed by a meta-model that prescribes its structure and organization, is used to specify the information object. This is critical to moving towards information model-driven architectures, where information objects are explicitly defined by a model that is decoupled from the software services. In this architecture, the domain model includes the dictionary of data elements along with their semantic relationships, which are captured either in a schema, or with more specificity in a complex model such as an ontology.

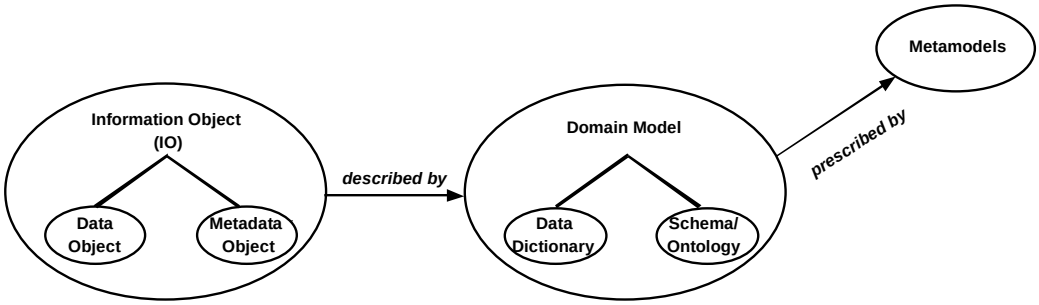


Figure 4-7: Relationship between Information Architecture Concepts

Figure 4-8 shows the relationship of information models to their associated objects, to a set of registry and repository services, accessed via a messaging middleware. This shows a set of domain information objects built on information services. As depicted, the model governs the structure and semantics of an information object. The object is registered within a registry service and stored in a repository service. Well-defined interfaces are used to support the process of storing metadata and data as described in section 3. Applications are then able to manage and interrogate the registry through well-defined interfaces. For applications that need to perform complex searches, that function is managed outside the registry. This allows leveraging of modern search-engine technologies which can handle semantic, free-text, and other search patterns.

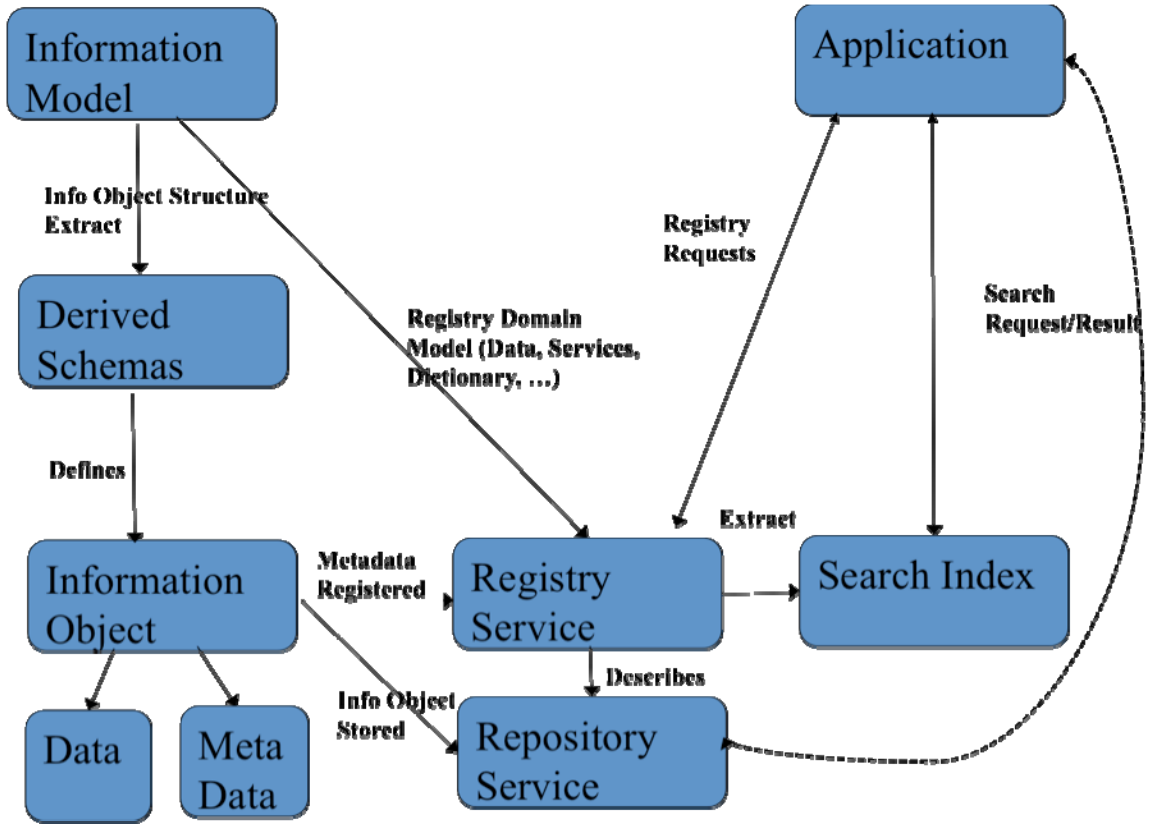


Figure 4-8: Relationship between Information Models and Services

5 SPACE DATA SYSTEMS PROJECTS

5.1 OVERVIEW

This section provides information about related space data system projects that could benefit from the use of the information architecture described in this document. The potential use of information architecture components in each project is summarized in table 5-1.

Table 5-1: Example Projects Using Related RASIM Concepts

Project	Information Architecture Concepts Used
Open Archival Information System (OAIS)	CCSDS reference model describing information objects, information packages, and archive service objects.
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 (EDS)	Uses concept of information objects, information packages, archive service object, registry service object for a national grid system.
National Virtual Observatory (NVO)	Uses concepts of information objects, information packages, archive service objects, and registry service objects for an international astrophysics interoperability effort.
PDS	Uses concepts of information objects, information packages, archive service object, and registry service objects for a national planetary science grid system within NASA.

5.2 OPEN ARCHIVAL INFORMATION SYSTEM

The CCSDS OAIS reference model (reference [5]) 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 information architecture. 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. With respect to information architecture, the OAIS DIPs, SIPs, and AIPs could all be considered information objects conforming to each respective package format specified in reference [5]. SIPs, DIPs, and AIPs all are logical structures that can be reified in many forms: XML files compressed using a Zip compression technique (e.g., XFDU), or binary-formatted data and metadata weaved together and compressed using a proprietary compression format (e.g., SFDU).

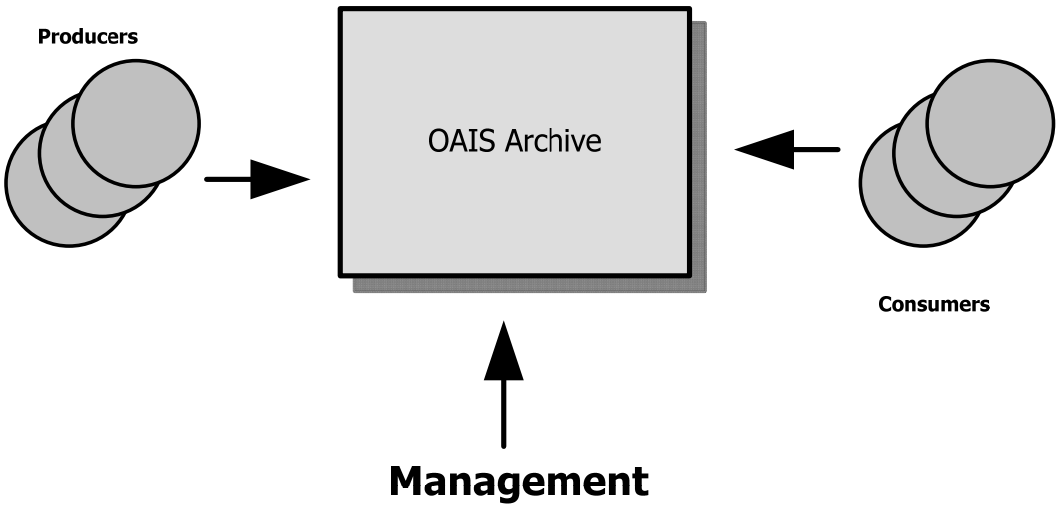


Figure 5-1: The OAIS Reference Model, Adapted from Reference [5]

OAIS-compliant archives are in the business of preserving, providing, managing, and collecting information. Inherently they are most related to the archive service object described in 3.3.5; however, since the OAIS reference model defines the standard data structures that an OAIS archive should use, which are all domain-specific instantiations of information objects, OAIS archives could utilize the information objects described in this document.

5.3 GRIDS

5.3.1 GENERAL

Recent work in the grid community (see reference [16]) has characterized a class of distributed data interoperable systems as *data grids* (references [8], [9], [23], and [26]). Data grids involve the identification of (different classes of) metadata (reference [26]), 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 each grid project uses the components of information architecture.

5.3.2 SPACEGRID

ESA's SpaceGRID Study [21] 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 a) distributed data management, b) data distribution, c) data access, and d) 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. Of the 146 requirements, a cross-section of design areas was identified, and user-desired requirements of grids were listed as the following:

- flexibility;
- portal;
- security;
- distributed access;
- human computer interface;
- virtual organization;
- collaborative environment;
- reliability.

Figure 5-2 depicts the proposed SpaceGRID infrastructure, which 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.

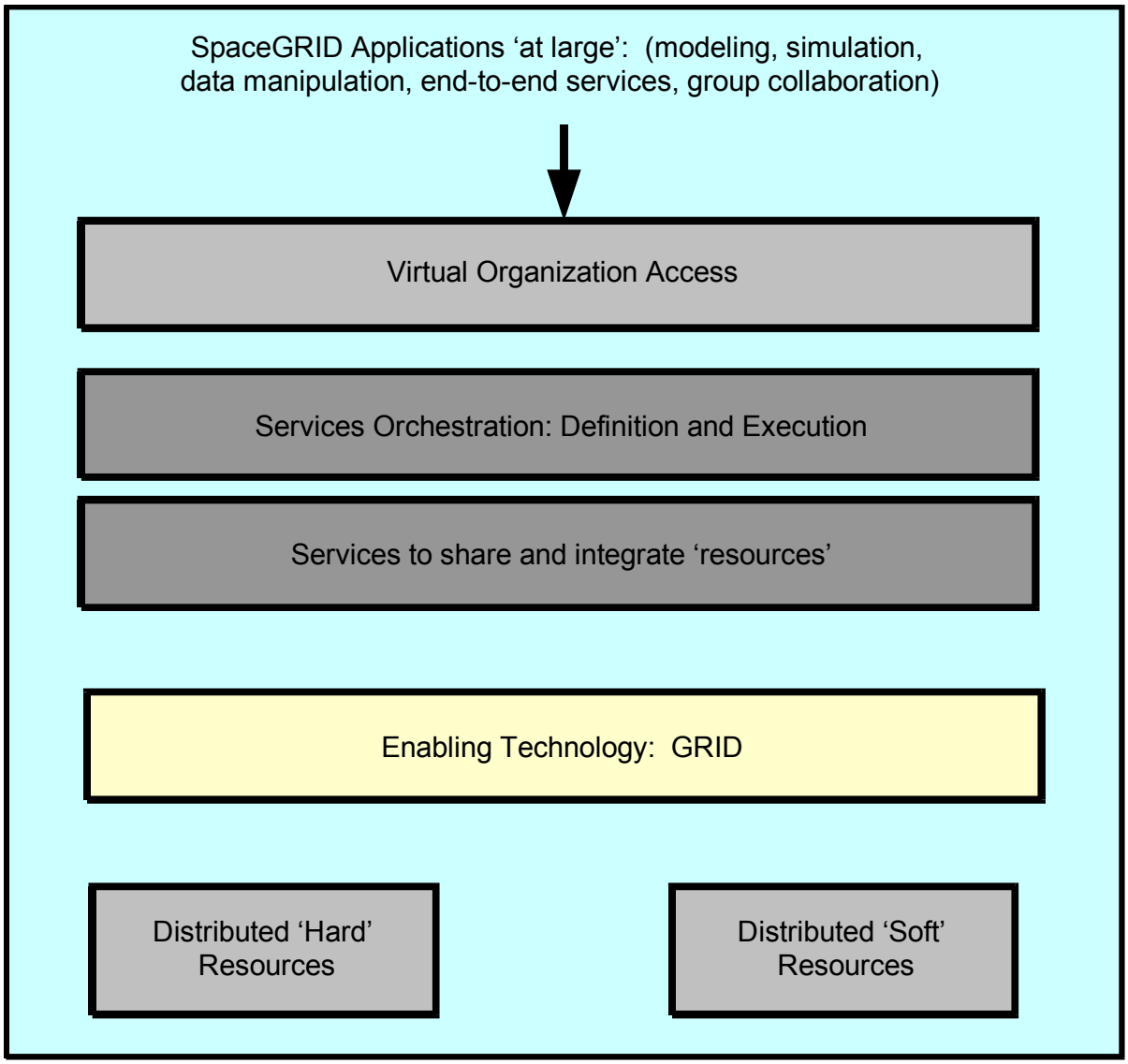


Figure 5-2: SpaceGRID Proposed Infrastructure

5.3.3 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, or 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 this document lies in the fact that EOSDIS is a domain-specific example of a) Earth science-specific information objects, b) Earth science meta-models, c) Earth science metadata objects, and d) Earth science domain models and ontologies.

5.3.4 EUROPEAN DATA GRID

The EDG is an EU- and ESA-funded project aimed at enabling access to geographically distributed data and computational resources (see reference [15]). 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 this document. 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 3, to query for and retrieve application information objects and information packages made available by the EDG system.

5.3.5 NATIONAL VIRTUAL OBSERVATORY

5.3.5.1 General

The NVO is an NSF-funded project whose goal is to enable science by greatly enhancing access to data and computational resources. NVO uses components from the Globus Toolkit (see references [16] and [20]) grid middleware infrastructure to distribute, process, retrieve, and search for astrophysical science data. The components of NVO are essentially the components described in this document: a) a well-defined information architecture, including information objects (or astrophysical data products), b) common models to describe the information objects, and c) software service objects (in the form of grid services) to exchange science data.

5.3.5.2 International Planetary Data Alliance

NASA's PDS and the ESA have recently completed an exploratory study (see reference [31]) and are working on an open protocol dubbed 'PDAP', or Planetary Data Access Protocol. The PDAP will directly aid in the standard exchange of information between the distributed, cross agency planetary science data. Planetary data exchanged via PDAP will use XML as a basis for data exchange. These data units will conform to the information architecture described in this document and the architecture of the system is quite similar to that of the existing PDS infrastructure (described in the following subsection).

5.3.6 PLANETARY DATA SYSTEM

The mission of the PDS is to efficiently collect, archive, and make accessible the digital data produced by or relevant to NASA's planetary missions, research programs, and data analysis programs. The PDS consists of seven 'discipline nodes' and an engineering and management node. Each node resides at a different U.S. university or government agency and is managed autonomously.

To address new demands, including increasing data volume, data complexity, and number of missions, the PDS is moving to a fully online, federated system that was designed using many of the information architectural principles presented in this document. Called 'PDS4', the system was designed to meet three fundamental goals: a) to support more efficient delivery of data from data providers to the PDS, b) to enable a stable, long-term usable planetary science data archive, and c) to enable computing services for data consumers to find, access, and use the data they require in contemporary data formats.

The PDS4 system architecture is comprised of common software components developed from registry and repository services compliant with the ebXML standard. The PDS4 information architecture, developed independent of and in parallel with the system architecture, involved knowledge acquisition across a large and diverse group of planetary science domain experts. The resulting information model consisted of a set of OAIS information objects captured in an ontology modeling tool and an ISO/IEC 11179-compliant data dictionary. The information model was used to generate an extensible set of XML schemas and associated documentation for use by the system, data providers, and data users.

6 RELATIONSHIP OF THIS DOCUMENT TO OTHER CCSDS STANDARDS EFFORTS

This document describes a Reference Architecture for Space Information Management (RASIM). RASIM describes functional components, and concepts of information management including metadata, data, and domain models. This includes the definition of canonical components for the management of information in any space data system. Clearly missing from the definition of RASIM in this document is the actual description of complex space data system architectural topologies and best practices of using software components, and information models described herein.

The future work of the CCSDS Information Architecture Working Group includes defining best practices and architectural designs and topologies constructed using RASIM using this document as the seminal guide for the initial description of such components. In its current state, this document serves to lay the grounds for at least two of the aforementioned CCSDS reference documents (one for best practices, the other for system designs), and potentially many more. For instance, it is clearly possible that each of the functional components described in this document (e.g., query service object, archive service object) deserve documents describing in detail their interfaces and interactions in their own rights.

NOTE – The work of the Information Architecture Working Group was halted because of a lack of agency resources to complete this work. As a result none of the other documents that are described here are scheduled for production at this time.

ANNEX A

APPLICABLE STANDARDS USED IN THIS DOCUMENT

A1 OVERVIEW

This annex details the applicable standards used in this document, such as the modeling notation standards, and the governing CCSDS standards that apply to this document.

A2 UML MODELING STANDARDS

A2.1 GENERAL

All of the UML figures drawn in this report are drawn using the UML 1.0 notation. UML was chosen because of its broad applicability and use in the design of modern software-intensive systems.

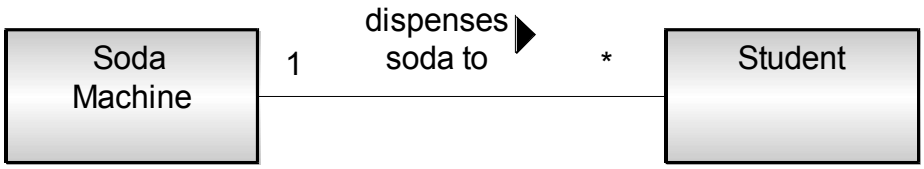
UML is used to represent data concepts in this document, along with software object concepts and relationships. The following UML diagrams were used extensively:

A2.2 UML CLASS DIAGRAMS

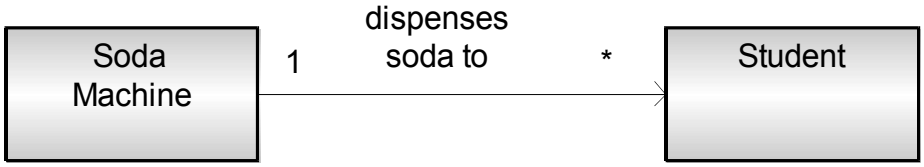
The UML Class diagram is used to show objects and relationships amongst objects. In particular UML defines *classes*, which are entities in a system that interact and interface with other entities. Classes can have attributes, interfaces, and relationships with other objects.

Primarily, the *association* relationship is used in this document. An association has a *name* and *direction*, which describes the name of the association (e.g., *has a*), and the direction (either uni- or bi-) from which the association originates and ends. Associations can have *roles* at each of their ends. Roles define how one end of an association will behave in certain situations. Associations can also have *cardinalities* at each end of the relationship, specifying how many of each class connected to each end of the association participate in the association. In this document, the cardinality **1** denotes a single participant from a class. The cardinality ***** denotes more than one, or *many* participants from a class.

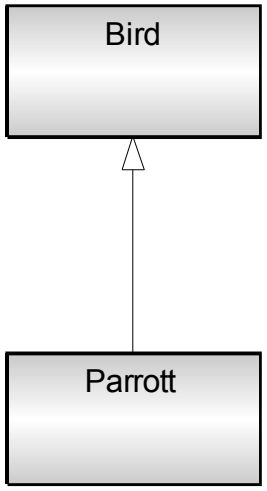
Using the example below, it is known that there is a *Soda Machine* class and a *Student* class, with an association between them with a name of *dispenses soda to*, which originates from the soda machine class and terminates at the student class. It is known that for one soda machine, there can be many students to whom it dispenses sodas. It should be noted that there are no roles defined in this example.



The association relationship shown above between soda machines and students can also be shown using the following notation:



There is one other type of relationship used in the UML modeling notation in this document to represent relationships between two objects, or classes. The type of relationship is a *generalization* relationship. Generalization relationships indicate that one class is a specialization of a more generic class, or that one class is a child of another class, which is its parent. The generalization relationship is usually shown with an unfilled arrow end at the end of an association line. In the example below, there is the parent class *Bird*, and a child class, *Parrott*, which is a type of bird; therefore it is a specialization of the more generic parent class. As such, there is a generalization relationship between the two classes.



Please see reference [1] and [2] for further clarifications on the UML modeling notation used in this document.

A3 CCSDS

Table A-1: CCSDS Information Standards Mapped to Information Architecture Concept

Information Architecture Concept	CCSDS Standard
Meta-Model Specification (section 2)	*Data Entity Dictionary Specification Language (DEDSL)* (CCSD0013) Blue Book. Issue 1. January 2002.
Archive Ingestion Model (section 3)	*Reference Model for an Open Archival Information System (OAIS)* Blue Book. Issue 1. January 2002.
Data Element Semantics and Specification (section 2)	*The Data Description Language EAST Specification (CCSD0010).* Blue Book. Issue 2. November 2000.
Specification of Application Information Object Format (section 2)	*Information Interchange Specification*
Data Value Representation (section 2)	*Parameter Value Language Specification (CCSD0006 and CCSD0008).* Blue Book. Issue 2. June 2000.
Packaging Specification (section 2)	*XML Formatted Data Unit (XFDU) Structure and Construction Rules.* White Book. Issue 2. September 2004.
Data Object Format Specification (section 2)	*Standard Formatted Data Units — Control Authority Data Structures.* Blue Book. Issue 1. November 1994.

This section presents a mapping of existing CCSDS Standards to the data and software components and ideas discussed in this document.

ANNEX B

ABBREVIATIONS AND ACRONYMS

aIMO	advanced information management object
AIO	application information object
AIP	archival information package
API	application programming interface
CFDP	CCSDS File Delivery Protocol
CM	content manager
DBMS	database management system
DDR	data description record
DEDSL	data entity dictionary specification language
DIP	dissemination information package
DO	data object
DSO	data store object
EAST	Enhanced Ada SubseT
ebXML	electronic business using eXtensible Markup Language
ECS	EOSDIS Core System
EDG	European Data Grid
EOS	Earth Observing System
EOSDIS	EOS Data and Information System
E-R	entity-relationship
ESA	European Space Agency
ESB	enterprise service bus
IMO	information management object

IO	information object
JMS	Java messaging service
MOF	meta-object facility
MVC	model-view controller
NSF	National Science Foundation
NVO	National Virtual Observatory
OAIS	open archival information system
ODL	Object Description Language
OMG	object management group
OODT	object-oriented data technology
OWL	Web ontology language
PDAP	Planetary Data Access Protocol
PDS	planetary data system
pIMOs	primitive information management objects
PVL	Parameter Value Language
QO	query object
RASDS	Reference Architecture for Space Data Systems
RASIM	Reference Architecture for Space Information Management
RDF	resource description framework
SGML	Standard Generalized Markup Language
SIP	submission information package
SLE	Space Link Extension
SM	service management
SOA	service-oriented architecture
SPASE	Space Physics Archive Search and Exchange

UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
URN	uniform resource name
W3C	World Wide Web Consortium
WS	Web service
WSDL	Web Services Description Language
WWW	World Wide Web
XFDU	XML Formatted Data Unit
XML	eXtensible Markup Language