N94-33799

A Reference Model For Scientific Information Interchange

Lou Reich
Computer Sciences Corporation
Code 502
4600 Powder Mill Road
Beltsville, Maryland 20705
Phone: (301) 572-8445
FAX: (301) 595-1774
lreich@gsfcmail.nasa.gov

Don Sawyer
Goddard Space Flight Center
Code 633
Greenbelt, Maryland 20771
Phone: (301) 286-2748
FAX: (301) 286-1771
sawyer@nssdca.gsfc.nasa.gov

Randy Davis
Laboratory for Atmospheric and Space Physics
University of Colorado
Campus Box 590
Boulder, Colorado 80309
Phone: (303) 492-6867
FAX: (303) 492-6444,
davis@aquila.colorado.edu

I. Introduction

This paper presents an overview of an Information Interchange Reference Model (IIRM) currently being developed by individuals participating in the Consultative Committee for Space Data Systems (CCSDS) Panel 2, the Planetary Data Systems (PDS), and the Committee on Earth Observing Satellites (CEOS). This is an ongoing research activity and is not an official position by these bodies.

This reference model provides a framework for describing and assessing current and proposed methodologies for information interchange within and among the space agencies. It is hoped that this model will improve interoperability between the various methodologies. As such, this model attempts to address key information interchange issues as seen by the producers and users of space-related data and to put them into a coherent framework.

Information is understood as the knowledge (e.g., the scientific content) represented by data. Therefore, concern is not primarily on mechanisms for transferring data from user to user [e.g., compact disk read-only memory (CD-ROM), wide-area networks, optical tape, and so forth] but on how information is encoded as data and how the information content is maintained with minimal loss or distortion during transmittal. The model assumes open systems, which means that the protocols or methods used should be fully described and the descriptions publicly available. Ideally these protocols are promoted by recognized standards organizations using processes that permit involvement by those most likely to be affected, thereby enhancing the protocol's stability and the likelihood of wide support.

II. Issues In Scientific Information Interchange

Figure 1 presents an overview of what is meant by information interchange. The left side indicates the existence of several pieces of information in various local forms and knowledge of the relationships among them. The objective for the data producer is to assemble these pieces and the appropriate knowledge in a way that can be transferred across a spatial and temporal gap to a consumer system where any or all of the pieces of information and the relationships between them can be identified, extracted, and used in processing and display.

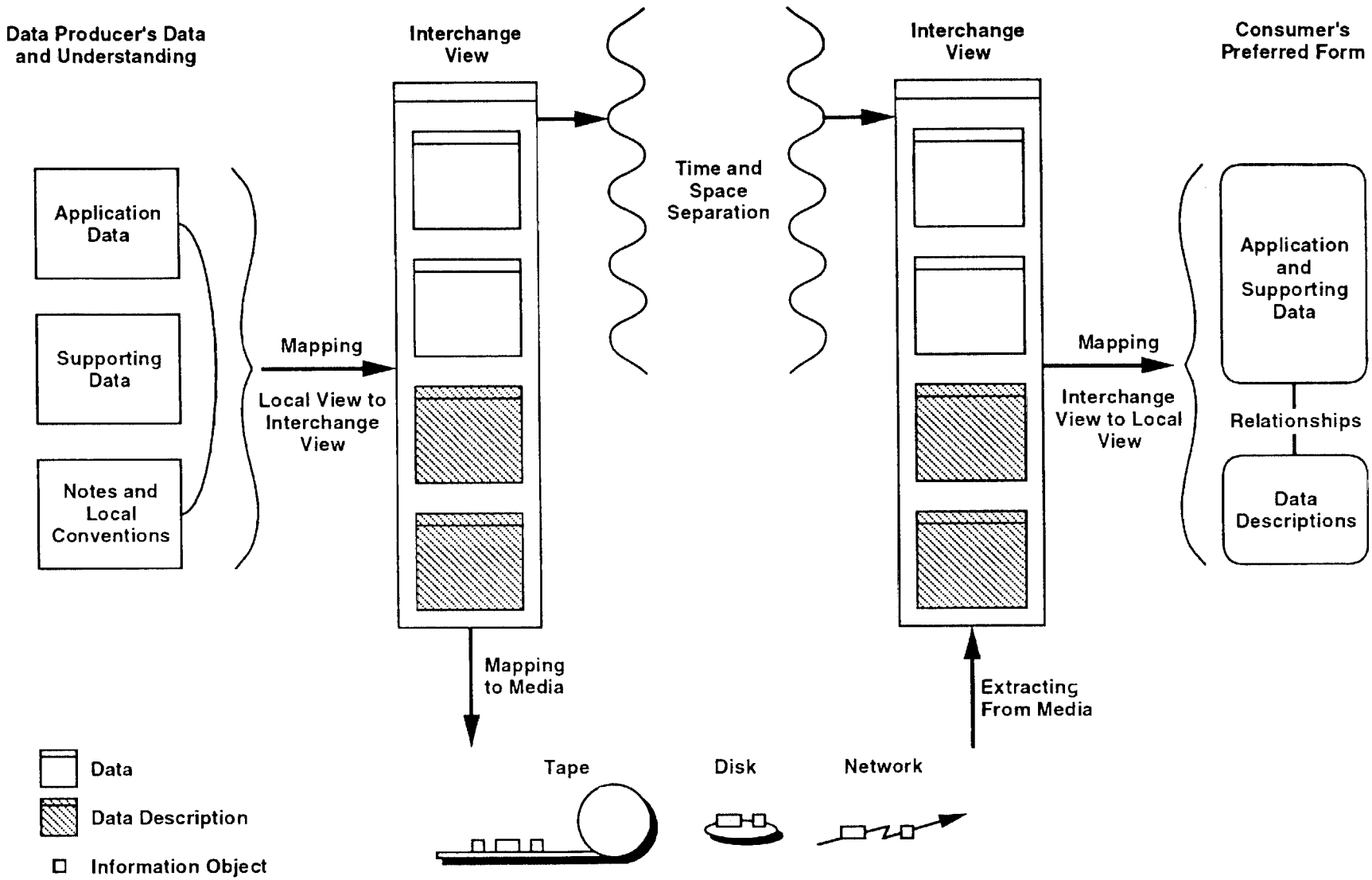

Figure 1. Information Interchange Process

An essential element in this view is the physical (spatial) separation of the two systems. Temporal separations can range from a fraction of a second to many decades.

The problem of moving strings of bytes reliably from senders to receivers has been successfully addressed by several suites of standards. The Open Systems Interconnection (OSI) model and the standards that adhere to it provide a solid framework for understanding and implementing systems that move data across networks. The packet telemetry and telecommand standards developed by the CCSDS supplement OSI-compliant protocols with capabilities specifically designed for communications with satellites. The Sony-Philips Red and Green Books provide the basis for encoding information on CD-ROM media so that the resulting disks can be read in any CD-ROM reader. These protocols can assure that a transmitted byte string is received completely and in the correct order (or if it is not, that the failure is reported to the receiver).

The protocols cited above do not, however, address all the needed aspects of encoding and interpreting the information within byte strings. They transport blocks, packets, and frames, whereas end users in the space sciences deal with images, spectra, tables, and maps. How then do we address the transport of the information objects, such as images and tables, to scientists? The OSI model allows for applications-level protocols that provide the rules for encoding and interpreting information within an applications domain. It is the applications-level protocols (sometimes with assistance from presentation layer protocols) that allow recipients to extract information from the bytes of data they receive. Few formally standardized data transfer methods for scientific information exist, but the need for them is growing. Most science disciplines within NASA are developing or seeking standard ways to transfer complex scientific information. The IIRM provides a mechanism for characterizing data transfer methods (with emphasis on those for scientific applications) so that users can describe the similarities and differences between existing or proposed methods. This may provide a basis for discussing the way individual science disciplines view their data and perhaps result in greater uniformity between data transfer methods for scientists. The more standardized the methods are the more automated the services can be for dealing with the information in both producer and consumer environments.

Several characteristics of space science applications complicate the information interchange process, including:

- Highly heterogeneous computing environments
- Voluminous data and metadata
- Wide variations in the level of user sophistication
- A large--and expanding--set of information relationships

The remainder of this paper discusses some of the key issues of information transfer in space science applications. This list is a first pass and is probably not comprehensive. Interested readers are encouraged to submit any additional issues or comments on current issues.

A. Encoding Information Into a Data Stream

Whenever information is stored or processed, it is encoded as a series of primitive data elements. Use of heterogeneous computer hardware and bit-efficient coding schemes for data from satellites and science instruments results in a wide variety of bit sequences to represent primitive data types (e.g., integer and floating point numbers). For efficiency, the data processed by a computer should be encoded in the formats that the computer hardware supports; however, these formats often differ for the computer systems used by the producer and consumer of a data stream. There are several ways to address this problem:

- The producer's system may know the local representation of the consumer's system and convert data to the consumer's local representation before sending

- The producer's system can inform the receiving system about the data representation and require the consumer to convert the data it receives to its local representation

- The producer's system can convert data into an agreed-on format, and the consumer's system can convert from the agreed format to its local representation

No single solution is best for all situations. Despite today's sophisticated and fast computer hardware, converting large volumes of information from one format to another for interchange or archiving is often impractical; data volumes appear to increase as rapidly as processing power. This means that a scientist's access to information may be limited simply by the difficulties of data translation.

Encoding issues also arise in every layer of software through which information must pass when transferring data streams. For example, some operating systems impose private record encoding schemes within files that can restrict or complicate the flow of data files within an open system.

Programming languages present an additional problem: a programming language's set of base types is usually richer than the primitive data types represented in hardware (for example, arrays and enumerated types). However, different programming languages use different conventions to encode the same base type, and encoding information as a sequence of base data types that can be recognized and manipulated by all the languages that might be used to process the information is often difficult. For example, exchanging arrays across different languages is often difficult because arrays for some languages are implicitly column major, and for others they are row major. These kinds of problems have led to specialized data definition languages (DDLs) that allow data to be fully described in a way that is independent of any particular programming language. Even with DDLs, some modification of the information may be required before the information is used with a specific programming language (the array majority issue is such a problem).

B. Identifying and Accessing the Information in a Data Stream

The receiver of a data stream must be able to locate, identify, and access each major information unit in the data. These units are called information objects; however, use of this term does not imply that the systems producing and consuming them necessarily conform to the principles of object-oriented programming.

For open systems, a very large number of different types of information objects may be transmitted. The producer knows the identity and order of objects within any data stream it transmits, but it is presumed that the consumer has no prior knowledge of the data contents. A mechanism is therefore required to identify and describe each information object in the data. The usual mechanism is to provide supporting information, or metadata, that identifies and describes the information objects. Some of the metadata acts like a table of contents or index in helping to locate and identify the information objects. Software is then provided to browse through a large set of information objects to find the specific objects required for an application or to create a useful subset of objects. Metadata are also used to describe the attributes of information objects and to describe the relationships between information objects.

Numerous issues are associated with metadata. First, the mechanism for encoding and supplying the metadata must be determined. Second, the amount and completeness of metadata needed to describe information objects and their relationships is inversely proportional to the inherent level of the consumers understanding of the information objects received. Producers must determine the metadata needed to make the transmitted data understandable and accessible to the intended audience. The requirements are particularly

stringent for archived data, where a data stream may be preserved beyond the life of any hardware or software that created it or that can access it. In such cases, sufficient descriptive information must be available to allow deciphering of the entire data stream.

Metadata are data, and like other kinds of data, generally require their own metadata to allow receivers to understand and interpret them. This meta-metadata must also be provided. A current mechanism for storing and providing some of this meta-metadata is a database called a Data Dictionary or Data Entity Dictionary (DED). The DED defines information in a consistent format.

C. Interpreting Information in a Data Stream

Received information must often be placed into a context broader than the containing data stream. A common problem is unambiguously identifying and naming an object so that it can be distinguished from all other information objects that exist in a large system. A traditional method of naming the information objects held in computer systems is by location, for example, directory path names for files. This method causes problems, however, when the location of an information object changes, then references to the object (e.g., a file reference appearing within a text document) must also change.

Another issue is how received information objects relate to other information objects within a large system. Software reusability depends in large measure on this issue, for if a piece of software has applicability to a wide variety of data objects, a mechanism is needed for determining which objects the software can and cannot handle. The inheritance mechanism used in object-oriented programming addresses this problem by providing a hierarchy or network to determine how each type of object is related to all other types. Typically in an object-oriented system, software that works for one type of object will work for all objects of that type and for all types of objects derived from the original type.

III. Overview Of An Information Interchange Reference Model

The IIRM consists of three layers, as shown in Figure 2. The layers support information partitioning and a degree of information hiding, which grows as one moves from the lowest layer to the top layer. This structure allows the functionality assigned to each layer to be addressed separately and allows users to assume that the functionality of the lower layers is provided in support of a given upper layer. An implementation need not adhere to strict information hiding to be consistent with this model; access to information at a lower layer may be needed to meet special circumstances. For a given implementation, the three layers work together. Note that not every implementation will interoperate with other implementations at the adjacent (lower or upper) layers.

The top layer of the IIRM is based on the object-oriented paradigm. This schema includes the definition of base types, a type hierarchy, and relationships that model the process of information interchange. Use of an object-oriented data model, by identifying the specific objects defined and supported (either implicitly or explicitly) by various information interchange methodologies, makes it possible to identify similar objects across implementations and to compare the capabilities and mechanisms of each implementation. This technique allows analysis of non-object-oriented methodologies through the identification of the implicit objects that a methodology supports. In addition, an object-oriented view allows for explicit definition of complex relationships among scientific data and metadata. Current object-oriented data models do not discuss underlying representation of data. Because such representation is an important aspect of science data exchange, the IIRM augments the object-oriented data model with the additional (lower) layers that deal with data representation issues.

The functionality addressed in each of the layers is described in the sections that follow.

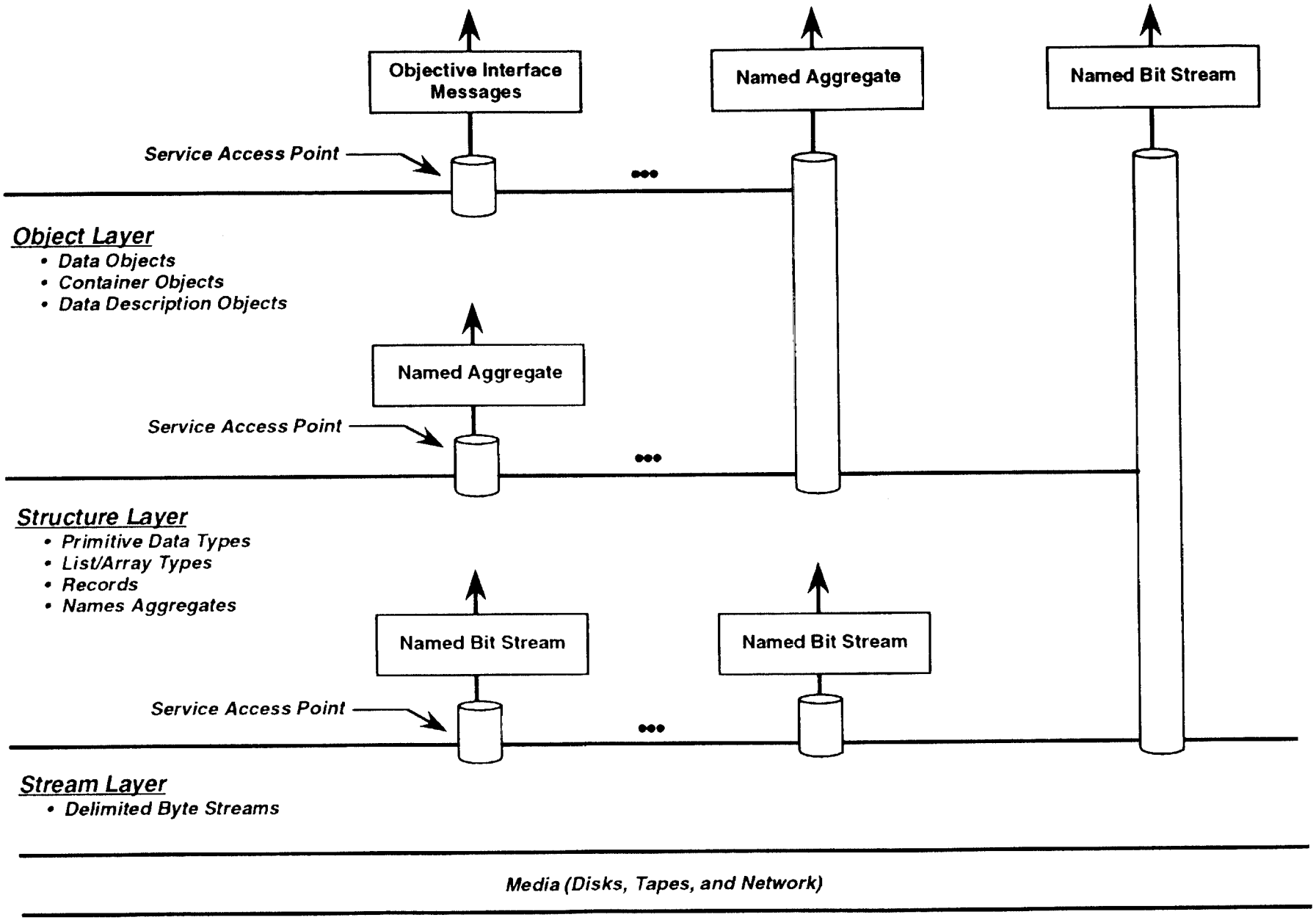

Figure 2. Information Interchange Core Model

A. Stream Layer

As noted previously, the IIRM augments existing models of the data transfer process, like the OSI model. Because the IIRM addresses issues found in the user-oriented top layers (the applications and presentation layers) of the OSI model, the IIRM can assume the existence of protocols for the lower five layers of the OSI stack (the physical through session layers) and need not duplicate the functionality of those lower layers. However, the IIRM applies not just for information interchange over networks; it is for information transported on media like tape and CD-ROM as well. The stream layer—the lowest layer of the IIRM—provides the interface between the IIRM and medium-dependent standards, protocols and mechanisms for data transport. It hides the unique characteristics of the transport medium by stripping any artifacts of the storage or transmission process (such as packet formats, block sizes, inter-record gaps, and error-correction codes) and it provides the higher levels of the IIRM with a consistent view of data that is independent of its medium. This common view is that data are simply collections of named sequences of bits. The term name here means any unique key for locating the data bytes of interest, including path names for files, a virtual channel ID for CCSDS telemetry, and so on.

Examples of standards and protocols that provide the functionality needed in the stream layer are ISO 9660 for CD-ROM, ISO standard labels on magnetic tapes, and file transfer protocol (FTP) on networks. For example, the ISO-9660 standard provides the volume and directory information needed to locate a file on a CD–ROM volume and sufficient information about the file format that a user retrieve the file as a sequence of bits. It ignores issues such as record structure (fixed length or variable length). The returned file is simply a sequence of bytes at this point; access to the information encoded within this file (or any other data stream) is addressed in the structure layer, described in the next section.

B. Structure Layer

As mentioned previously, information must be coded into primitive data types that can be recognized and accessed by computer hardware and operating systems. In the structure layer, information is viewed as a sequence of primitive data types. For any implementation, the structure layer defines the primitive types that are recognized. This usually means at least characters and integer and real numbers. Primitive types can also include the aggregation types typically supported in computer languages, including the array (where each element consists of the same type of data) and a record or structure that can (potentially) hold more than one type of data. An enumeration type is also often provided as a primitive type. As noted earlier, because of the efficiency constraints often imposed on space science data, users sometimes create their own representations for primitive data types (e.g., 6-bit integer numbers). Issues relating to the representation of primitive data types are resolved in this layer.

All types of information are built from these primitive types. Through the structure layer, the information is mapped into primitive types and then into the corresponding bits and bytes of a data stream. Note that a single structure may be distributed among several streams. The issues of the structure layer are often thought of as data format issues and are handled automatically by DDLs.

C. Object Layer

The highest layer in the IIRM is the object layer, wherein information is represented as objects that are recognizable and meaningful to end users. For scientists, this includes objects such as images, spectra, and histograms. The object layer adds semantic meaning to the data treated by the lower layers of the model. Some specific functions of this layer include the following:

- Recognizing data types based on information content rather than on the representation of those data at the structure layer. For example, many different kinds of objects—images, maps, and tables—can be implemented at the structure level using arrays.

Within the object layer, images, maps, and tables are recognized and treated as distinct types of information.

- Presenting applications with a consistent interface to similar kinds of information objects, regardless of their underlying representations.

- Providing a schema mechanism to identify the characteristics of objects that are visible to users along with the relationships between objects.

To characterize information in the object layer, the IIRM uses concepts and terminology that have been developed in the object-oriented community. Agreement is not unanimous about what constitutes an object-oriented approach, but most models of object-oriented systems currently in use or in development share the key features needed. One such model, the concrete object model developed by the Object Data Management Group (ODMG), is being used to facilitate the standardization of Object Database Management Systems (ODBMSs). This paper uses the ODMG's approach to describe the entities at the object layer of the IIRM. This model can be briefly summarized as follows:

- The basic modeling primitive is the object. As with real-world objects, information objects can be arbitrarily complex. For example, in the real world, both a bolt and an automobile are objects, although the latter is significantly more elaborate than the former. Similarly, a pixel of an image, an entire image, and the entire dataset containing the image can all be treated as objects.

- Objects can be categorized into types.

- Instances of objects are created using object types as templates. Each object instance possesses all the characteristics of its type. The set of all instances of a specific object type is called that type's extent.

A type has one interface and one or more implementations. The interface defines the external public behavior supported by all instances of a type. The components of the interface are as follows:

- Attributes—Characteristics of the object for which an external user can get the values for any instance of the object

- Relationships—Logical paths an external user can traverse to move from an object instance to related object instances

- Operations—Actions an external user can invoke on an instance of an object

An implementation defines the internal or private data structures and procedures that support the externally visible states and behaviors. A single interface may have several alternative implementations.

Object types are related to one another using the supertype/subtype (or parent/child) relationship. This relationship links all object types according to their shared characteristics and is commonly represented as an acyclic graph. For example, a type called Faculty Member may have subtypes called Instructor and Associate Professor, and Faculty Member may in turn be a subtype of Person. All of the attributes, relationships, and operations defined for a supertype are inherited by the subtype. The subtype may add attributes, relationships, and operations to introduce behaviors or states unique to the instances of the subtype. A subtype may also refine the attributes, relationships, and operations it inherits to specialize them to the behavior and range of state values appropriate for instances of the subtype.

IV. Model Schema For Scientific Information Interchange

The three-layer model just described is general and can describe many data interchange problems. The goal of the IIRM, however, is to have a model specifically suited to describing scientific data interchange. In this section the model adds a domain-specific object-layer schema that allows characterization and comparison of systems for scientific data interchange.

To show what the description of an object looks like, Figure 3 presents a formal description of an image as represented in the object layer of a hypothetical data system. The descriptions of each component are given in plain English, although for a real data system the descriptions of attributes, operations, and languages will typically be in a formal, computer-readable language.

A key point about scientific data in general can be found in the description of relationships in the sample: Manipulation of a primary scientific data object such as an image frequently requires substantial auxiliary data. For example, interpretation of image objects requires a knowledge of the camera detector calibration as well as geometric information—orbit position, spacecraft inertial attitude, and the mounting and pointing of the camera on the spacecraft. These kinds of information may be of scientific interest in their own right (for example, the trajectory of a spacecraft reveals something about the number, position, and masses of objects in the solar system), but if in a scientific application they are primarily used to analyze of other information objects such as images and spectra, these kinds of information are auxiliary data. Auxiliary data can be collected into a set of objects. The attributes, operations, and relationships for each type of auxiliary data object are highly dependent on the object's role in data analysis. With orbit/attitude/pointing information, for example, there may be attributes that indicate the inertial frame of reference (e.g., ecliptic and equinox of date) and there may be operations to return spacecraft position at a specific time.

Another key point arises from the requirement that the IIRM be applicable to an open system environment. In such an environment, it should be possible to devise software that can receive and manipulate new types of objects with little or no reprogramming To do so such software must have access to the metadata that describes the interface to each new object. A database of interface definitions for objects is sometimes called an Object Interface Repository (OIR) or an Object

Dictionary (OD); these are specific cases of a DED. Such a DED can identify the interface components—attributes, operations and relationships—for the known types of objects. The DED can also provide a formal definition of each of these components. A DED and the definitions within it can be considered objects called metadata objects. Transferring metadata objects from one DED to another or from a DED to an end user may require that the metadata objects be encapsulated for transport other kinds of objects, so that metadata objects may exist outside of the framework of a DED.

Given the complexities of scientific data, typical data requests may require the transfer of several types of primary objects (for example, some images and their associated image-intensity histograms), along with associated auxiliary objects, such as calibration files and orbit/attitude/pointing data, and metadata objects that describe each of these other kinds of objects. Thus mechanisms must be available for collecting other kinds of objects and encapsulating them during transport; such mechanisms are called container objects. Container objects may contain their own kinds of metadata: for example, they may provide a sort of table of contents that identifies and locates each object within a container.

Figure 4 provides a preliminary class hierarchy. Each downward arrow indicates a subtype relationship. For example, both Container Object and Data Object are subtypes of Object and they inherit all the methods of Object.

When applying the IIRM in the analysis of a data system or a data interchange methodology, seek to identify the types of objects that are used by the system. Examples of this analysis are given in the next section. Some data systems can be best described by modeling from the top (i.e., object layer) down, whereas others are better suited for modeling from the bottom (i.e., stream layer) up. Either a top-down or bottom-up approach may be used when applying the IIRM model.

Object Type	**Image**
Description	An image represents a mapping of the intensity of electromagnetic radiation in two or three spatial dimensions. Digital images consist of a set of picture elements, or pixels, with the value of each pixel proportional to the intensity of light measured by the camera system within the areal extent of the pixel.
Supertype	Image is derived from type Array, which describes homogeneous multi-dimension data structures. Type Array is in turn a subtype of the most basic type called Object
Subtype	Subtypes of this type can be created to characterize images taken by specific camera systems.
Attributes	The following are the attributes—the visible characteristics—of images: • Number of dimensions (2 or 3) in the image [positive integer numbers] • Number of pixels in each dimension [positive integer number] • Number of bits per pixel [positive integer number] • Content [character string] • Time that picture was taken [date/time] • Exposure time [time] • Wavelength or frequency range [real numbers] • etc.
Operations	The following are the operations that can be performed on all images. These augment the set of operations that are inherited from the parent type Array.
Subsample	Create a new image consisting of a contiguous set of the pixels from an image.
Average	Create a new image by averaging a specified number of contiguous pixels from an image.
Generate Histogram	Create a Histogram object for which each element is the total number of pixels within an image with a given intensity value.
Relationships	The following are relationships involving image objects:
Calibration	This relationship relates an image to a characterization of the sensor that took the image.
Pointing	This relationship relates an image to where the camera is pointing.

Figure 3. Sample Type Description of an Image

V. Applying The Reference Model

In this section, the IIRM is used to characterize current data exchange methodologies as follows:

1. Identify the primary object types defined by the methodology at the object layer, along with the auxiliary, metadata, and container objects used.

2. Identify the primitive data types defined in the structure layer and the way the object-layer entities map to the primitive types in the structure layer

3. Identify the media and data exchange mechanisms supported at the stream layer.

The following data interchange methodologies are described here:

- Hierarchical Data Format (HDF)
- Planetary Data System (PDS)
- Standard Formatted Data Unit (SFDU)

Figure 5 summarizes the key characteristics of these methodologies.

A. Hierarchical Data Format

The HDF was created by the National Center for Supercomputing Applications (NCSA) to provide access to common types of scientific data. An HDF is a self-describing file format that contains a set of tagged objects. NCSA provides a comprehensive library of routines in C and FORTRAN to create and to retrieve data from HDF files. In addition, there is a sizable body of applications software, both public domain and commercial, for accessing data in HDF format.

HDF has been selected as the baseline standard data format for the Earth Observing System Data and Information System (EOSDIS). Consequently, the HDF data model is undergoing significant evolution to provide high-level data types commonly used by scientists to model Earth-related phenomena. The following analysis is based on Version 3.3 of HDF, released in September 1993.

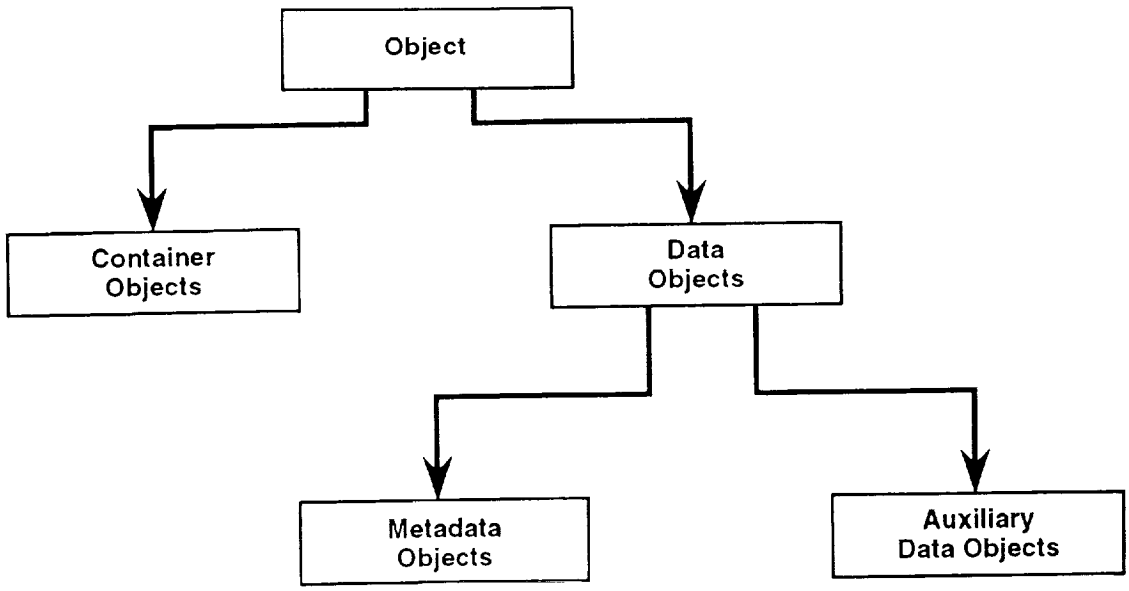

Figure 4. IIRM Object Layer Class Hierarchy (Preliminary)

	PDS	HDF	SFDU
Stream Layer	• Requires file structure • Uses FTP/DECNET or disk structure	• Requires direct access file structure	• Allows any level of service that supports conversion of bits to bytes
Structure Layer	• ODL labeled objects • Machine dependent datatypes, IEEE datatypes	• Tagged record structure • Machine dependent datatypes, IEEE datatypes	• Stream of Label-Value Objects • Data Definition Language allows wide specification of primitive types and "record structures"
Object Layer	• Limited class hierarchy • No methods defined other than attribute retrieval • Data Objects – Images – Histograms – Spectra – Tables • Container Objects – Files – Volumes • Metadata Objects – Catalog – Data Entity Dictionary • Auxiliary Data Objects – SPICE Kernals – Gazeteer Objects	• No current class hierarchy • Formal Application Program Interface (API) for each data type • Data objects – Raster Images – Palette – Multidimensional Array (SDS) – Tables (Vdata) • Container Objects – Vgroups – Files • Metadata Objects – Annotation – Attributes with SDS	• No current class hierarchy • No methods defined other than object insertion/retrieval from containers • Data Objects – Application Data Objects – Supplementary Data Objects • Container Objects – Exchange Data Units – Application Data Units – Description Data Units • Metadata – Data Description Packages – Data Entity Dictionary Objects – Catalog Attribute Objects • Auxiliary Data Objects – Supplementary Data Objects

Figure 5. Preliminary Descriptions of HDF, PDS, and SFDU Using IIRM

Object Layer

HDF provides a set of Application Program Interfaces (APIs) through which all application data access must occur. The primary data objects within HDF are classified by the relevant API. These APIs are equivalent to defining the external interface (i.e. operations and relationships) of objects at the IIRM object layer in that they are independent of the internal implementation of the objects within HDF files. The APIs currently defined are:

- Raster Image API: Allows the user to store and access raster images and optional color palettes. Three optional forms of image compression are supported: JPEG, run-length encoding and IMCOMP compression.
- Palette API: Defines color tables for 8-bit raster image data.
- Scientific Data Set (SDS) API: Allows the storage and access of multidimensional arrays with specific attribute data. The interface provides the ability to slice an array and work with the resulting subset of the data.
- NetCDF API: Also allows storage and retrieval of multidimensional arrays. This API supports the netCDF data model, developed by the Unidata program of the University Corporation for Atmospheric Research, which is a richer data model than SDS. Additional features include an "unlimited" dimension and global and local attributes.
- Vdata API: Allows storage and retrieval of collections of data that can be viewed as record structures. This includes data meshes, polygonal data with connection information, packed data records, and sparse matrices.
- Vgroup API: Allows general hierarchical grouping of HDF objects.
- Annotation API: Allows labels and unstructured text to be associated with any HDF object or with an entire HDF file.

HDF does not support the concept of type hierarchies and formal inheritance. NCSA's commitment to backward compatibility with previous versions of HDF has led to some features that would probably be implemented differently if the system had been engineered to be object-oriented from the outset. For example, the NetCDF API is a pure superset of the SDS API, since these two APIs developed separately, the relationship between the SDS and NetCDF is not a true subclass/superclass relationship.

Structure Layer

The structure layer in HDF supports a standard set of primitive data types including real numbers (IEEE floating point), integer numbers (unsigned and signed 2's compliment), and character strings (big-endian byte ordering). In addition, HDF can store the machine-specific representation of reals, integers, and character strings for supported platforms.

The basic building block of an HDF file is the data object, which contains both data and information about the data. A data object has two parts: a 12-byte data descriptor (DD) and a data element. Figure 6 below illustrates two data objects.

A DD has four fields: a 16-bit tag, a 16-bit reference number, a 32-bit data offset, and a 32-bit data length. The tag of a DD tells what kind of data is contained in the corresponding data element. A tag and its associated reference number uniquely identify a data element within an HDF file.

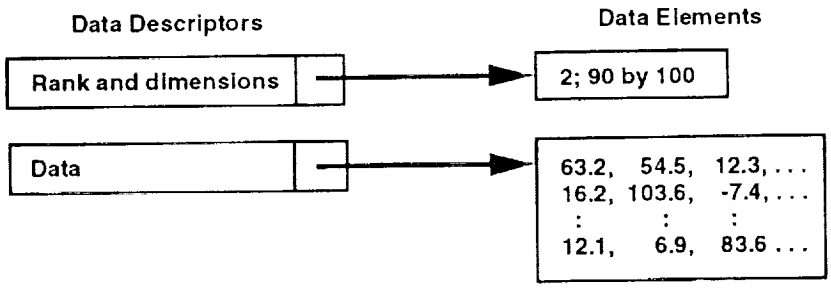

Figure 6. Two HDF Data Objects

DDs are stored in a linked list of blocks called data descriptor blocks, or DD blocks. The file header, DD blocks, and data elements appear in an HDF file in the following order:

- File header
- First DD block
- Data elements
- Additional DD blocks and data elements

Stream Layer

HDF depends on a stream layer that provides direct access capabilities. The tagged structure in the structure layer requires efficient seeking to specific locations in a single HDF file. HDF files may be stored or transmitted on sequential media, but they must be moved to direct access media before they are accessed.

B. Planetary Data System

The PDS acquires, archives, and distributes much of the data that NASA collects on bodies in this solar system other than Earth, including planets, comets, and asteroids. When the prototype of the PDS began in 1983, it inherited substantial amounts of existing planetary science data in many different formats. It was not practical to reformat all of those data into a standard representation. therefore, the PDS developed a methodology for describing data in a way that both human users and computers could identify and understand the content of a data file or stream. This methodology describes data objects that are set forth in a language called the Object Description Language (ODL). A label (typically called a PDS label) encoded in ODL is attached to every data file or data stream that flows into or out of the PDS to identify the objects in the file or stream. Gradually the PDS evolved a relatively comprehensive set of standard objects and data providers are encouraged, even required, to submit data in a format that is consistent with the standard objects definitions. The standard objects are defined through the Planetary Science Data Dictionary (PSDD).

Object Layer

PDS object model is still in development and the description below includes some new facets to the model that are currently being adopted and formalized through the PSDD.

Primary Objects

The two simplest types of objects, called Element and Bit Element, can hold a single instance of a primitive data type. The two are similar, but the Bit Element type can handle primitive data that are not aligned on byte boundaries. There are two general aggregation objects—Array and Collection that hold element objects. An array is homogeneous—all elements must have the same underlying primitive data type—while the collection can be heterogeneous, which makes it analogous to the record or structure data type found in many data models.

The PDS also provides several primary data objects that are specialized for space science applications. These include:

- Histogram
- Image
- Table
- Spectrum

PDS does not use the inheritance mechanism to define subtypes of these objects. Instead, each of these object classes provides all the attributes needed to describe nearly all instances of the object. For example, all images are objects of type Image. Figure 7 describes the image object.

Three aspects of the PDS object model, as illustrated above for images, deserve elaboration. First, there are only a few PDS objects that have formal subtypes. Specifically, there are several important subtypes of the Table object, including a Palette object to hold color table information for image display and a Series object to hold time series (or similarly organized) data.

Second, no currently no formal operations defined for images or any other type of PDS object exist. There are several reasons for this omission, including the difficulty in agreeing on what the standard operations should be and neither the PSDD nor the ODL used for PDS labels currently have the syntax or semantics necessary to describe operations. A unique problem with defining standard operations arises when PDS object types like Image are designed to cover a vast extent of object instances, with no use of subtyping to provide specialization. This means that some PDS object types are so complex that there is no single piece of software that can account for all the possible permutations of their optional attributes. For example, no single piece of software can handle all instances of PDS images.

Third, there are no formal relationships defined for PDS objects, except for the limited use of supertype/subtype as noted above and a simple relationship called Contains indicates an object holds other types of objects. The most notable example of the Contains relationship is the Table object, which contains one or more Column type objects. In general, if two or more instances of PDS objects are related—for example, an image and its associated histogram together within a file—this relationship is only implicitly indicated by the objects that are contained within the same file and described together by the same PDS label.

Auxiliary Objects

The planetary community has developed a standard representation for orbit/attitude and pointing auxiliary data. This standard is called SPICE, where the letters of the acronym stand for the kinds of information that are handled: spacecraft, planets, instruments, coordinates,

and events. The Navigation and Ancillary Information Facility (NAIF) at the Jet Propulsion Laboratory (JPL) provides auxiliary data to projects in SPICE format. The NAIF also maintains the SPICE standard and provides an extensive Fortran library of operations to support SPICE-encoded data. SPICE files (called SPICE kernels) are considered to be PDS objects and their attributes are defined through the PSDD.

Object Type:	**Image**
Description:	An image represents a mapping of the intensity of electromagnetic radiation in two or three spatial dimensions. Digital images consist of a set of picture elements, or pixels, with the value of each pixel proportional to the intensity of light measured by the camera system within the area extent of the pixel.
Supertype:	PDS has no formal inheritance mechanism, hence there is no formal supertype for type Image.
Subtype:	There are no formal subtypes since there is no formal inheritance mechanism. In practice there are numerous subtypes of images, since the standard image format produced by each of the cameras abroad a planetary spacecraft can be considered to be a subtype of type Image
Attributes:	The following attributes are mandatory and must appear in each description of an image object instance: • Lines—number of scan lines in image • Line_Samples—number of scan lines in image • Sample_Type—Type of primitive data that makes up a pixel of the image • Sample_Bits—The length of a pixel. There are also a large number of optional attributes which may or may not appear in a description for an image object instance, depending upon whether or not they are needed (if omitted, they each have a default value). A representative set of the optional attributes for Image are given below: There are also a large number of optional attributes that may or may not appear in a description for an image object instance, depending on whether or not they are needed (if omitted, they each have a default value). A representative set of the optional attributes for Image are given below: • Bands—The number of spectral bands in an image • Band_Storage_Type—Method used to interleave spectral bands in a multi-spectral image • Encoding_Type—The method used to compress an image, if any • Line_Prefix_Bytes—The number of bytes at the beginning of a scan line that contain non-image data (for example, gain information or timing data) • Line_Suffix_Bytes—The number of bytes at the end of a scan line that contain non-image data
Operations:	The PDS does not formally define operations upon objects.
Relationships:	There are no formal relationships defined for Image objects.

Figure 7. Description of PDS Image Object Type

Another type of PDS auxiliary data is the Gazeteer object, which is a subtype of the Table object that provides information about geographical features on planets and satellites. For example, it provides the name of a feature or region, the body on which it is found, and its coordinates on the body.

Metadata Objects

The PDS defines a set of metadata object classes called Catalog Objects. They are used primarily to provide a template for data providers who are supplying information to be placed into the PDS catalog of data holdings. Some catalog objects are also used to augment the standard attributes of data objects. A prime example is the Map Projection catalog object, which provides a set of attributes that define a map projection. Frequently the raw images from planetary spacecraft are processed by mapping their pixels onto a standard map projection grid. When an object of this kind is created, a Map Projection catalog object is placed within the Image object in a PDS label to describe the map characteristics of the data. Users can correlate each pixel of the image with its location on the planet from information from the Map Projection object.

Container Objects

The PDS has several objects that serve to collect other objects. The most important is the File object, since most PDS data are transferred within files. Since much of the data that the PDS distributes is on volume-oriented media like CD-ROM, there is also a Volume object to provide information on the organization of a collection of files.

PDS container objects often have their own metadata. There is a Header object, which defines the headers that in turn describe the contents of data files. Aside from the standard PDS labels, this includes the VICAR labels found on many planetary images and the FITS headers found on many planetary datasets derived from observations with earth-based telescopes.

Structure Layer

The PDS has a fairly ordinary set of primitive scalar types: character strings, integer, and real numbers, enumeration types. It also uses the CCSDS format dates and times, allowing these to be considered primitive types as well.

There is no single required representation for primitive types. It is the instantiation of a primitive type as an Element type object, or as a component of some other kind of object (like a pixel of an image), that determines its format. Thus primitive types like numeric values can be represented in nearly any computer's native format. The PDS label that describes a data object provides information on the encoding of the primitive data types within the object. For example, a PDS label will identify whether or not the real number values that make up a histogram object are encoded in VAX format, IEEE format, or another type of format.

There is no separate data definition language for PDS-labelled data, because the PDS labels contain information needed to understand the structure layer, A PDS label does not as a rule provide a complete structure layer mapping: it does not rigorously establish the position of every data item in the object. Users have to rely upon numerous implicit rules to map from the PDS label's description of objects to the underlying representation of those objects within the structure layer.

Stream Layer

Small amounts of data are sometime provided to users over the NASA Science Internet. Typically FTP or DECNET file copy is used to transfer files over the network. Larger quantities of data are typically provided to users on CD-ROM. There are many CD-ROM titles that adhere to PDS standards. These disks adhere to the ISO-9660 standard. There are currently no specific stream layer services provided by the PDS to access data files in a way that is transparent of the medium of transport.

C. Standard Formatted Data Units

The CCSDS Panel 2 has been developing, adapting, and adopting standards to improve information interchange within and among space agencies. CCSDS standard recommendations have been developed in support of a methodology called SFDUs. Briefly, this methodology involves the association of a small label with a collection of data values, forming a labeled value object (LVO), and the incorporation within the label of a globally unique identifier (i.e., Authority and Description Identifier, or ADID) of a description of the data values. This description may be a CCSDS Panel 2 standard and thus be found in a formal CCSDS recommendation document, or it may be defined by a user and be found at a Control Authority Office (CAO) set up by a participating agency conforming to the CCSDS standard titled "Control Authority Procedures." The primary function of a CAO is to register, archive, and disseminate data descriptions in response to user requests. These descriptions may themselves be composed of several labeled objects, including a formal (computer interpretable) description of the format of the data values, a text description of the mission and instrumentation involved in the creation of the data values, and software that may be used to obtain particular services from the data values. As such, these description LVOs may also be packaged with the data LVOs to form a self-describing data package.

Stream Layer

The SFDU standards assume the existence of stream layer services such as those provided by the volume/directory file system on a CD-ROM, the sequence of files on an ISO/ANSI standard labeled magnetic tape, and FTP for network file transfer. The provision of a sequential byte (8-bit) stream is the minimum requirement of the SFDU standards, while the use of named (e.g., directory/file names) byte streams permits the construction of sequences of labeled data objects that cross multiple files on random access media. This functionality is described in the Structure Layer.

Structure Layer

The standard titled "SFDU Structure and Construction Rules" is the primary CCSDS Panel 2 standard that interfaces with stream layer services. It defines an SFDU 20-byte label to support three primary functions:

1. Provide mechanisms to determine the end of a sequence of data values (i.e., encapsulate the data values) associated with the label

2. Provide a code which gives a general classification (e.g., data, data description package, supplementary data) to the encapsulated data values

3. Provide a globally unique identifier of a description (e.g., data description package) of the encapsulated data values. It also defines a number of standard descriptions and assigns globally unique 8-character standard identifiers (e.g., "CCSD0001") to them.

Application of this standard to the stream layer converts the byte stream view into a view of a sequence of hierarchically organized labeled value objects. This sequence may span multiple files on both sequential and random access media. One or more such sequences may be defined on a physical volume, or within a single file. There is no explicit provision for crossing multiple physical volumes with a single sequence, but it is possible if this is supported by the stream layer. It should be noted that the standard can be applied in such a way that many files are not required to contain labels. Thus the standard can also be applied to pre-existing data streams and to files conforming to other standards.

The labeled value objects at the lowest level of the hierarchy have a content that appears as a sequence of bytes from the stream layer. The structure layer function of interpreting this sequence of bytes into a sequence of primitive datatypes (e.g., integers, characters, and reals) is accomplished by interpretation of the Data Description Record (DDR) found within the Data

Description Package (DDP) identified in the label. This linkage of information is illustrated in Figure 8.

The DDR can be expressed in a number of standard languages that have been documented in CCSDS standards. Currently these include "ASCII Encoded English (CCSD0002)", "Parameter Value Language (CCSD0006)", and the draft standard "Enhanced Ada Subset (EAST)." The level of language-related automated support for access to the labeled value object depends on the language selected and ranges from presentation (e.g., ASCII/English) of a text description of the record structure(s) within the value to full parsing of record structures (e.g., EAST). Alternative support may be obtained from software associated with the particular ADID. This software may be provided as an additional object within the DDP.

DDPs are archived in a CAO so that any DDPs not present in the data stream may be obtained from the CAO. DDPs are expected to provide a complete description of the values whose labels contain their ADID, and in addition to the DDR which supports the structure layer function, they are likely to include a DED object and other semantics which may be used to support object layer services as described in the next section.

Object Layer

The SFDU standards provide a very general mechanism for representing and transmitting data objects. The SFDU standards do not currently provide a fully object-oriented approach: there is no class hierarchy; nor are methods defined, other than services for insertion and retrieval of data from containers. But SFDUs can be used to encapsulate data objects complete with their attributes and methods. SFDUs also provide container objects for combining collections of primary objects with the auxiliary data and metadata needed to interpret them. Thus the SFDU concept is one of a very few data interchange mechanisms that are designed to encapsulate and transmit all of the kinds of information contained in a scientific data system, whether object-oriented or not.

Primary Objects

Unlike the PDS and HDF methodologies described above, there are no specific primary data objects in the SFDU concept. Instead the SFDU standards provide a general object class called an Application Data Object (ADO). (Each SFDU object class has a one-letter identifier and an ADO is also called an I class object. As described in the structure layer discussion, the ADID in the label points to a DDP that fully describes the LVO. The Data Entity Dictionary (DED) with the DDP gives all the attribute names for the LVO type. In the future the DED will also contain relationship information about the LVO type. The DED is further described later in this section. For example, a scientist can use the ADID of an ADO to determine whether the data in the SFDU is an image, map, spectrum, or whatever, and to tell whether the object is the FITS format, PDS format, or some other format.

Auxiliary Objects

Since the SFDU standards have been developed with scientific applications in mind, there is a specific class of SFDU called the Supplementary Data Object (SDO) (or S class) that is used to contain auxiliary data. For example, if a spectrum is transferred in an ADO the calibration information for the spectrum can be placed into a SDO and the S class supplemental SFDU can then be transferred with the I class SFDU that holds the spectrum. As with ADOs a SDO may contain virtually any kind of data in any format desired, and the ADID for the SDO provides the key to determining the content and format of the object.

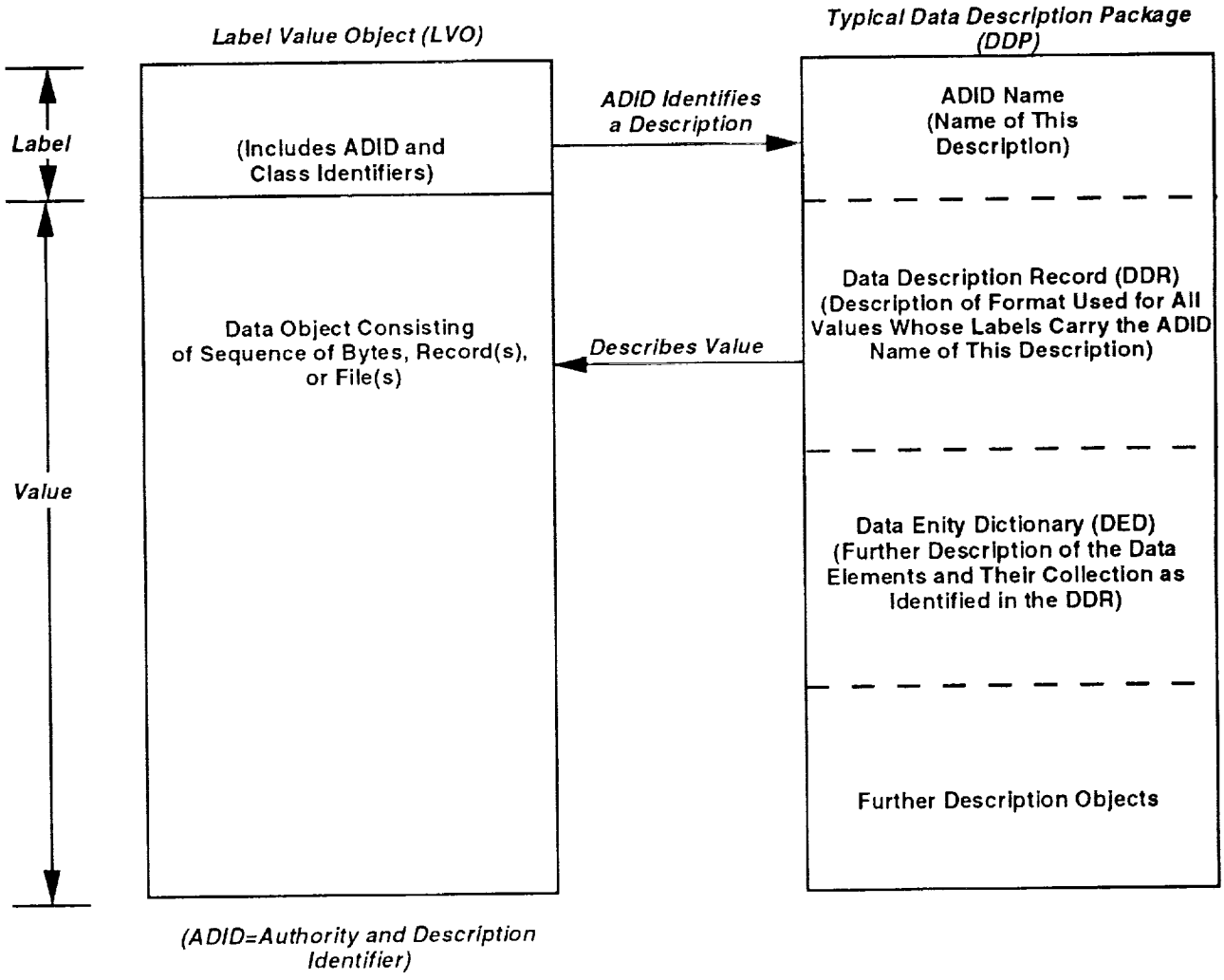

Figure 8. SFDU Label Value Object and Its Description

Metadata Objects

An important aspect of the SFDU concept is the ability to encapsulate metadata as well as data. There are three types of metadata objects defined by the SFDU standards:

- DDO (or D class)—These objects are used to hold the data descriptions that map an SFDU object—for example, an ADO—into the structure layer. The definition is given in a DDL. A DDO provides the mapping for a specific instance of an SFDU object. For example, a DDO may provide the data definition for a specific data table. Other data tables may have very different representations and hence would have their own DDO to describe them.

- DED Objects (or E class)—These objects are used to hold descriptions from a DED. The descriptions define types of objects rather than specific object instances. They can also define the terms used in object type definitions. For example, if an object has an attribute called Length, a DED object can specify the minimum and maximum values allowed for Length. The CCSDS is currently completing work on a standard representation for the information within DED objects. This standard representation uses the Parameter Value Language (PVL) to encode the DED information.

- Catalog Attribute Object (CAO) (or K class)—Data systems often maintain a catalog—a database that describes the data held within the system. The CAO can be used to transfer information to and from a catalog or a similar database. When a data system transfers applications data to a user it will often provide the pertinent catalog information or other attributes for the transferred data objects. The CAO supports this by holding the attributes of a set of ADO wrapped within a container SFDU. As with other types of SFDUs, the form and content of a CAO are not constrained by the SFDU standards. The information might be given in tabular format, where the columns are the attributes of the objects that are being described and each row of the table contains all the attributes for one data object. Alternatively, catalog attribute information can be given using PVL or a similar keyword/value notation, where there is a keyword/value pair for each attribute of each object.

Container Objects

The SFDU methodology provides three types of container objects:

- Exchange Data Units (or Z class)—These objects are the most general encapsulation mechanism for SFDUs. An Exchange Data Unit (EDU) can hold essentially any combination of the SFDU objects described in this section, including other EDUs.

- Applications Data Units (or U class)—These container objects can be used to aggregate a set of related ADOs and SDOs. An Applications Data Unit (ADU) may include a CAO that describes the other objects in the container. An ADU can also hold other ADUs.

- Description Data Units (or F class)—These container objects can be used to aggregate DDO, DED Objects, and any other metadata objects.

VI. Relationships With Other Reference Models

This section provides a comparison of the IIRM and two other models: the IEEE mass storage system reference model and the familiar OSI reference model for communications.

A. IEEE Mass Storage System Reference Model

Information on the Mass Storage System (MSS) Reference Model (RM) was obtained from the paper "Mass Storage System Reference Model: Version 4", which was published in the

proceedings of the Goddard Conference on Mass Storage Systems and Technologies, Volume 1, 1992.

The MSS RM establishes a client server environment to provide access to a (potentially) distributed system that accepts and returns named Bitfiles. This storage model addresses data interchange over time (i.e., storage), but not over space (i.e., an instance of a MSS is not moved to a new location). In contrast, the IIRM addresses data interchange over both time and space. Since data moved over time and space may end up stored in a MSS, it is useful to perform a mapping between the IIRM and the MSS RM.

The MSS RM named Bitfiles appear to be virtually identical to the named bit streams that the IIRM Stream Layer provides to the Structure Layer. The one exception is that the MSS RM Bitfiles also have a set of attributes such as file creation date, file owner, etc. Such attributes have not been called out explicitly in the IIRM, although they must exist and be accessible to the Structure and Object Layers. In other words, the entire MSS RM addresses functionality covered in the IIRM Stream Layer.

B. ISO Open Systems Interconnect Reference Model

The ISO OSI RM addresses the interchange of information over time and space using electronic networks. In contrast, the IIRM applies to both networks and physical media as interchange mechanisms.

The OSI model is a seven-layer model, which makes use of the information hiding principle of layers. The functionality of layers one through five (Physical through Session Layers) is to establish a connection between two communicating nodes and effect the transfer of data bits between them. This is similar to the functionality of the IIRM Stream Layer, although the name capability associated with this bit stream as output from the Session Layer appears to depend on the particular protocol standards defined for this layer.

The sixth layer of the OSI model, called the Presentation Layer, is intended to convert a bit stream into recognizable data types. While it is hard to determine from the OSI model itself the extent of this functionality, a clearer picture emerges from an examination of the ASN.1 protocol defined for this layer. For this layer, the functionality is similar to the IIRM Structure Layer, which includes the identification of common data types, and their aggregation into named structures.

The seventh, and top, layer of the OSI model, called the Application Layer, is intended to provide user applications with a number of common services. The types of services to be provided, as shown by some of the protocols defined for this layer, include electronic mail, a directory service, and a file transfer service. There is considerable parallel with the IIRM Object Layer, as these layers are intended to provide user applications with a service view of the underlying data structures. Differences include the object orientation of this layer in the IIRM (although an object view of the Application Layer should be possible) and the IIRM focus on understanding scientific data by focusing on identifying objects of scientific interest. The fact that the OSI model addresses network functionality leads to identifying Application Layer services for what are highly common network service needs (e.g., electronic mail). The types of objects (and their services) being addressed by the IIRM Object Layer could, in principle through standardization, enter an expanded OSI Application Layer.

The OSI Application Layer file transfer service, differs from the IIRM file transfers that are handled within the Stream Layer. This is not a contradiction to the mapping between the models just described. The functionality requested from a file system in the IIRM is to provide named bit streams. The functionality provided by an FTAM file transfer in the OSI Application Layer includes the recognition of common data types. The IIRM views the recognition of data types, and the provision of services from them, are more usefully obtained from an object view, not from a file view. Mechanisms that take this object view could use an FTAM service, in principle, in either of two ways: 1) by not using the capability of ASN.1 to

describe the data types, and instead describing the file content as a bit string, thereby reducing FTAM to simply providing named bit streams, or 2) by using FTAM to include the functionality of the IIRM Structure Layer, and then providing an object view of the FTAM file content. These variations in mapping reflect options on the level of services requested, and the ways they may be combined.

VII. Summary And Future Plans

The IIRM provides a basis for comparing data systems and data interchange methodologies at three levels: as represented by a stream of bits (the stream layer); as a stream of primitive elements (the structure layer); and as a collection of objects. By applying this model similarities and differences can be called out in the systems that are used for scientific data interchange and data analysis. The object layer of the model is unique as it accounts for primary scientific data like images and spectra that require auxiliary data for interpretation, metadata for description, and containers for encapsulation. The IIRM allows the user to describe how all these elements fit together for a specific data system or application.

In the future the IIRM will be refined and the model applied to data interchange systems other than the three that were analyzed in this paper. This analysis should permit data system designers and implementers to improve the compatibility and uniformity of information interchange where practical. This may, for example, make it possible for a scientist to compare spectra of the Earth's atmosphere with those from other planets, even though the spectra may be retrieved from different data systems in quite different formats. Capabilities like these will be especially important if we want to reduce the burden on scientists from dealing with the form rather than the content of scientific data.