NASA Report to NARA on OAIS Based Federated Registry/Repository Research: May 2005-January 2006

Don Sawyer	*Lou Reich*	*John Garrett*	*Sergey Nikhinson*
Donald.Sawyer@ gsfc.nasa.gov	*LReich@ csc.com*	*John.Garrett@ gsfc.nasa.gov*	*SNikhins@ csc.com*

March, 2006

Table of Contents

1 Introduction

The National Aeronautics and Space Administration's Goddard Space Flight Center through its Space Sciences and Exploration Directorate's National Space Science Data Center is performing research into advanced information encapsulation, information models and procedures, and highly scalable ingest mechanisms based on the Open Archival Information System Reference Model (ISO 14721:2003) (1) and the emerging XML Formatted Data Unit (XDFU) technologies for contributions supporting NARA's requirements to provide the American public with access to federal, presidential, and congressional electronic records collections.

This research is being conducted in coordination with standardization activities under the Consultative Committee for Space Data Systems (CCSDS), but is not limited to those activities. It benefits from the efforts of other agencies participating in the standardization work addressing the XML based packaging of data and the development of formal mechanisms for the submission of data to archives. It also applies the emerging standards to NASA and NARA specific data and ingest requirements to determine the utility of the draft standards and to illuminate both technical and operational issues.

It is widely recognized that techniques for packaging data and supporting metadata into logical or physical containers provide useful mechanisms for a variety of situations within and external to archives. Researchers are going beyond the simple creation of tar files to incorporate standardized types of metadata playing various roles. One of the earliest efforts in this direction was the development of the ISO standard 12175 (2), known as the Standard Formatted Data Unit (SFDU), developed under the auspices of the CCSDS. In July, 2000, the World Wide Web Consortium released the "Report on XML Packaging (3)". This effort outlined the requirements, issues and potential solutions to the problem of packaging XML metadata and binary into a single file. The W3C membership felt that this was an important issue, but it was not a high priority for typical XML user and did not start a Working Group. They did issue the results of the study and urged interested parties to create a single standard.

The CCSDS recognized the need to develop a new generation of Information Packaging standards to meet the new requirements including use of the internet as the primary data transfer mechanism, leveraging the better understanding of long-term preservation from the OAIS RM, and incorporating XML as an emerging universal Data Description Language. The Information Packaging and Registries (IPR) working group was chartered in the fall of 2001 to develop recommendations in this area.

CCSDS prefers to adopt or adapt an existing standard rather than start from scratch to meet identified requirements. So after the development of scenarios and requirements, the IPR WG evaluated existing technologies and alternative solutions prior to any XFDU development. The efforts studied were METS developed under a Digital Library Federation initiative, Open Office XML File Format developed by SUN and other members of the Open Office Consortium, the MPEG-21 efforts in ISO, and the IMS Content Packaging Standard developed by the IMS Global Learning Consortium. There was significant discussion on adopting the METS standard but the focus on digital libraries datatypes and the lack of a clear mapping from the METS metadata to the OAIS RM led to the decision to use the flexible data/metadata linkage from METS but to implement an independent XFDU mechanism.

Leading up to this research support, several versions of the CCSDS Draft Recommendation**,** **"**XML Formatted Data Unit (XFDU) Structure and Construction Rules" and the XFDU Toolkit Library, a reference implementation consisting of a set of JAVA Libraries and a partial GUI had been developed. In late 2004 a stable version of the Recommendation was approved for review by the CCSDS Engineering Standards Board and prototyping and testbed efforts were initiated by CCSDS agencies.

The CCSDS has also established the Data Archive Ingest (DAI) Working Group. It has produced a standard, which has also become an ISO standard, called the Producer-Archive Interface Methodology Abstract Standard (4). Informed by the OAIS reference model, it provides a model for negotiation between the Producer and the Archive, and it includes many steps leading to a Submission Agreement and a formal model of the data to be submitted. The current primary task of the DAI Working Group is to develop an implementable mechanism for the formal model that describes the organization of data to be delivered to an archive, and it must work with a standard delivery package structure. This will be referred to as the Submission Information Package (SIP) Model standard. Its primary focus has been on the metadata needed for the description as it plans to make use of the XFDU standard for delivery of the data.

In this report we describe progress in the development of the two standards, including rationale for certain decisions that have been made to date. We also describe recent experience in testing the XFDU standards in the context of software implementability and for use in packaging existing data from NASA and from NARA. We finish with summary of findings addressing what has been learned and noting some known issues.

2 Current View of the Draft Standards

2.1 XFDU Draft Standard and Reference Implementation (May, 2005)

The main purpose of the standard (5) is to define a specification for the packaging of data and metadata, including software, into a single package (e.g. file or message) to facilitate information transfer and archiving.

While the primary CCSDS scope of application is the entire space informatics domain from operational messaging to interfacing and working with science archives in the context of the OAIS Reference Model, its applicability should be much broader because of the commonality of issues with other domains and specifically the needs of the NARA for receipt and management of digital records.

A high level view of the XFDU is shown in Figure 1. It consists of an interchange file, called the Package Interchange File, that contains an XML structured Manifest Document and possibly other files. However it also logically includes other external objects (typically files) pointed to from within the Manifest Document. The Package Interchange File may also be an XML file, or it may be some type of binary file archive format such as ZIP or JAR.

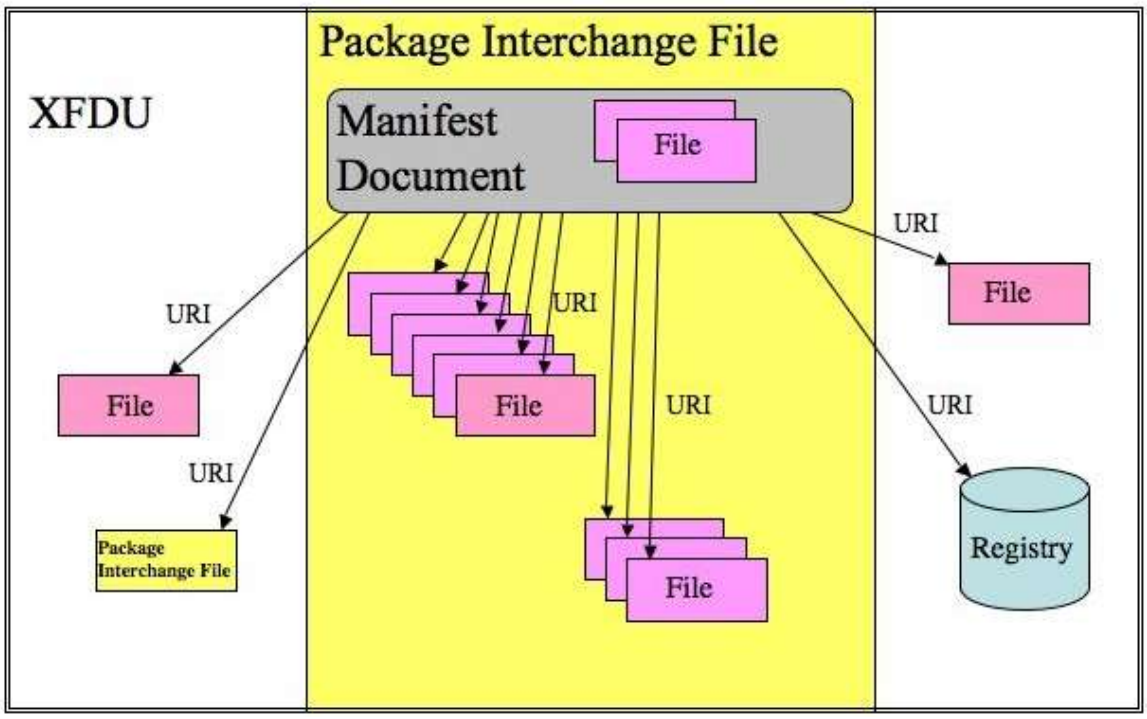

Figure 1. An XFDU consists of an interchange file containing a Manifest Document and possibly other files, but logically includes other external objects (files) pointed to from the Manifest Document.

The Manifest Document has a few major components whose functions and relationships are shown in Figure 2.

The Manifest includes a mandatory structure map section, called an Information Package Map that defines one or more Content Units.

A Content Unit may contain other Content Units, and each Content Unit has a number of other optional attributes and elements. Its attributes allow reference to associated Metadata Objects by internal pointers to elements in the Metadata Object section. Although not shown explicitly in Figure 2, several of these attributes may be used to categorize the referenced Metadata Object distinguishing among Representation Information, Preservation Description Information (PDI), and Descriptive Information as defined in the OAIS reference model. Content Unit elements may include other Content Units, may be internal pointers to elements in the Data Object section or may be external pointers to other XFDUs. Therefore a Content Unit can be used to associate a Data Object with one or more Metadata Objects, and multiple Content Units can present a hierarchical view of these data/metadata associations.

The attributes of a Data Object in the Data Object section are used to provide information such as mime type, size in bytes, checksum value and type, an internal pointer to associated Representation Information, and an identifier for information registered with a given registration authority. The elements describing a Data Object enable the object to be described as a sequence of one or more byte streams. The location of each byte stream is given either by a pointer (e.g., URL), or it may be included as a Base64 octet sequence. Note that each byte stream also has a set of attributes that can be used to provide mime type, size in bytes, checksum value and type. Further, each byte stream may have an associated transformation element giving the type of transformation that has been applied to the byte stream.

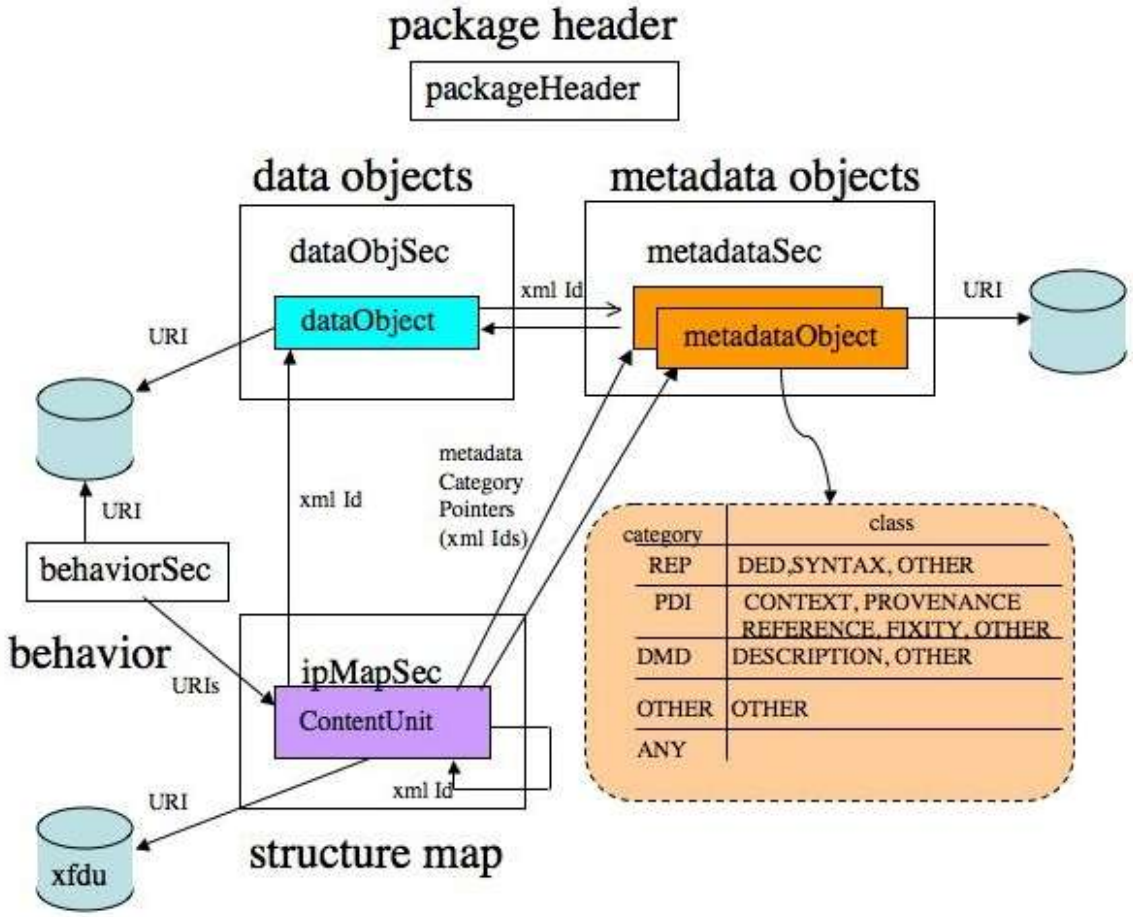

Figure 2. Manifest Document includes a structure map section that gives a view of one or more Content Units. Each Content Unit can point to a data object and one or more associated metadata objects. Each metadata object may be classified in a number of ways.

In the Metadata Object section, the attributes of a Metadata Object, like those of a Content Unit, can be used to categorize and classify the objects, including the ability to distinguish among Representation Information, Preservation Description Information (PDI), and Descriptive Information as shown in Figure 2. The elements describing Metadata Objects are use to either encapsulate the actual object in base64, or to point to a Data Object in the Data Object section. This allows a Metadata Object to also be described as Data Object in the Data Objects section. Since this description includes an attribute that is an internal pointer to Representation Information, a Metadata Object can be associated with its own Representation Information. Note that this mechanism allows the construction of OAIS defined 'Representation Nets' when the associated Representation Metadata Objects are also held as Data Objects.

A Behavior Section contains one or more Behavior Objects (behaviorObject) that associate executable behaviors with content in the XFDU object. A Behavior Object contains an Interface Definition (interfaceDef) that represents an abstract definition of the set of behaviors represented by a particular Behavior Object. A Behavior Object also may contain a Mechanism that is a module of executable code that implements and runs the behaviors defined abstractly by the interface definition.

A Package Header contains administrative metadata for the whole XFDU Package, such as version, operating system, hardware, author, etc, and it may contain metadata about transformations and XFDU versions /extensions that must be understood to successfully process the contents of the XFDU. An example of this metadata is a reference to an implementation of an algorithm to reverse a transformation that has been applied to some of the data objects within the containing XFDU

2.1.1 The XFDU Toolkit Library

An XFDU reference implementation API library using JAVA has also been generated. It conceptually consists of two layers. The first layer is a low level API representing each structure in the XFDU schema. The second layer is a higher level API that aggregates part of the functionality from the first layer. This allows easier access to constructing and manipulating an XFDU package. Both layers can be used either individually or in combination to manipulate a package, however the higher layer API doesn't provide all the functionality of the lower layer API. This means that while it can be used, for example, to create and populate a package with major pieces of information, one still needs to invoke the lower layer's methods to deal with numerous optional attributes of the XFDU elements. It is expected that coverage of the higher layer will grow over time to cover more areas of XFDU packaging.

2.2 Producer-Archive Interface Specification (PAIS) Draft Standard

The key objective of the PAIS standard (6) is to provide a method to formally define the digital information objects, along with their important inter-relationships, that are to be transferred by an information Producer to an Archive. Another objective is to support the effective transfer of these objects in the form of Submission Information Packages (SIPs) as modeled in the OAIS reference model. (Due to this objective, the PAIS Draft Standard is sometimes referred to as the SIP Standard.) If these objectives are met, use of the standard should facilitate validation by the archive that all the objects expected have been received and that they conform to the characteristics expected.

While the primary participants in the development of this standard are members of various space agencies, it is expected that this standard should have much wider applicability.

A high level view of the process involving use of this standard is given in Figures 3 and 4 as extracted from the draft document (6). The Producer is assumed to have an understanding of the type of data objects to be transferred, and by using one or more Descriptor Models as provided by the Archive, is able to create Descriptor instances corresponding to each type of data object (typically one or a few files) that is to be transferred. Descriptor instances include attributes identifying the descriptor type, number of data objects of this type to be transferred (if known), title for this type of data object, identification of the parent Descriptor instance, and identification of the SIP model to use in transferring the data object. There may also be various optional attributes taken from the standard, such as relationships to other Descriptor instances, or other attributes defined as needed for the Archive Project. Collections of such data objects and even collections of collections would each have a Descriptor instance defined. This modeling includes categorizing the Descriptors as relating to either ‘data objects’ (DOs), ‘complementary data objects’ (CDOs), or ‘collections of DOs or CDOs.’ The DOs are viewed as the primary data of interest, while the CDOs are viewed as relating to and supporting the DOs. The collections are, of course, just grouping of these data objects. The set of resulting Descriptor instances will form

a hierarchical structure called a Plan of Transfer (POT). This is intended to be iterated between the Producer and the Archive and should give the Archive the ability, in principle, to do validation on the incoming data objects to whatever level of specificity has been achieved via the POT.

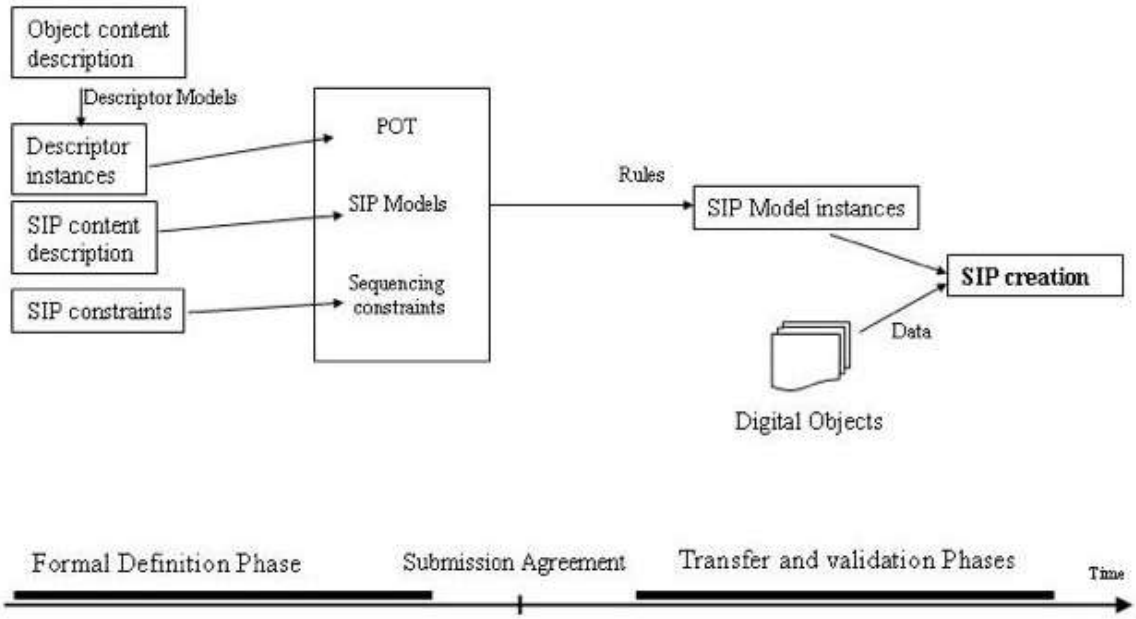

Figure 3: The process starts with the development of Descriptor Instances based on the data objects to be transferred and on the standardized Descriptor models. A Plan of Transfer is created along with models of how to map the data objects into types of SIPs for transfer. During the transfer phase, the data objects are instantiated into actual SIPs by the Producer using the SIP model constraints.

The document also currently specifies an abstract view of a SIP model in terms of attributes that are to be incorporated into a given SIP instance. Each SIP model must identify the Descriptor instances associated with the data objects to be transferred with this particular type of SIP. It must also provide a mapping from the identification of the Descriptor instance to the individual file names associated with that Descriptor instance, however this is not required until the SIP instance is generated. A given Archive Project may need multiple SIP models if there is a need to have different sets of Descriptor instances associated with different SIPs. There may also be a need for constraints on the delivery sequence of SIPs and data objects. The draft standard recognizes this need with a related set of attributes.

The SIP model instances, combined with the actual data objects for transfer, are then ready to be packaged into a container. This mapping to an underlyling container mechanism is under development. There will be a mapping to the XFDU standard within this specification, most likely in the next version of the document. The result of using the mapping is the creation of a SIP package, as shown in Figure 3.

It is assumed that the SIP package is transferred to the Archive successfully. As shown in Figure 4, the Archive receives the SIP and begins SIP validation by looking into the SIP, recognizing the SIP type and associated identifiers of Descriptor instances, and using the Descriptor instance information for comparison with the data objects found in the corresponding section of the SIP

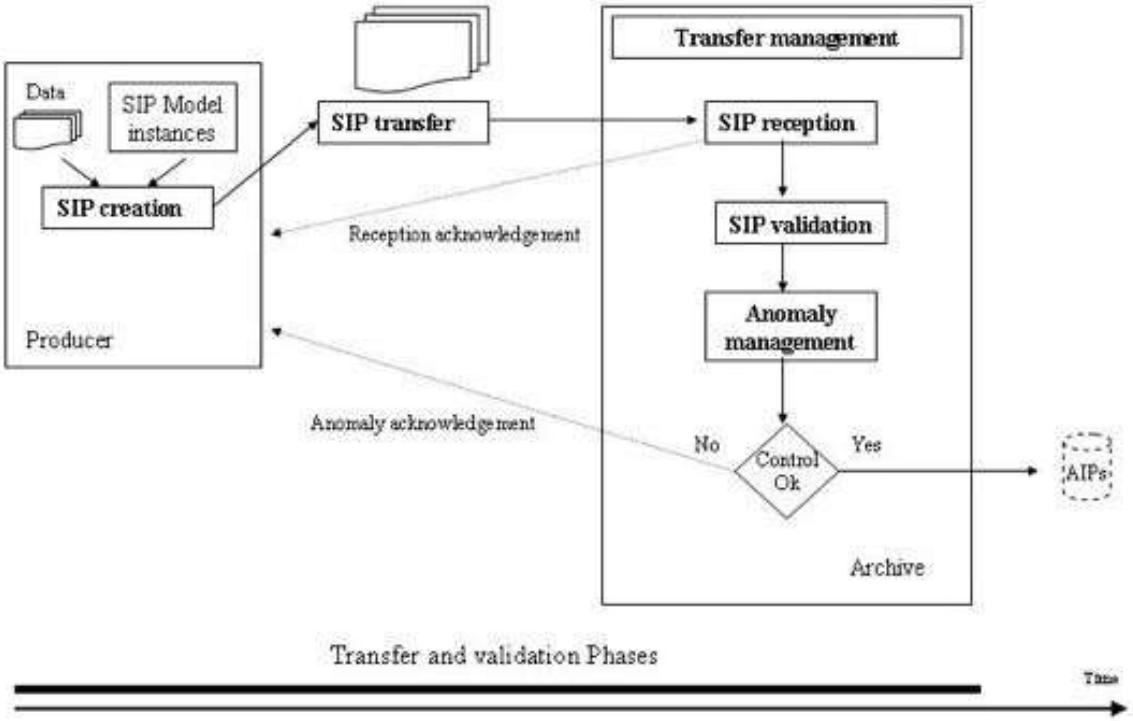

Figure 4. Created SIPs are transferred to the archive where they are validated against the Plan of Transfer and the SIP models previously agreed between Producer and Archive. Acknowledgements and anomalies are noted.

Any anomalies are noted and result in further interaction with the Producer. Otherwise the SIP is found acceptable for completion of the Archive ingest process and the production of Archival Information Packages (AIPs) for preservation.

The current document includes an XML schema for the specification of a generic Descriptor Model. It will eventually include an XML schema for the specification of a generic SIP Model, and it will include a mapping to the XFDU standard.

This document also addresses steps in the validation process, anomaly management, and the management of modification to the POT while the Archive Project is underway. It remains to be seen if any of this will be included in the final standard as it may be found to be over specified for most Archives.

3 Major XFDU Activities during Research Period

During the CCSDS IPR workshop in April, 2005, there were many comments (both editorial and technical) from NASA and other agencies on the XFDU draft standard. Most of the issues involved XML Naming Conventions and minor schema clarification and were resolved at the meeting.

The unresolved technical issues involved the XFDU approach to XML Schema Specialization and XML Schema Versioning. The analysis during the meeting identified the need to define Best Practices in these areas and the definition of an XML Schema repository service for XFDU XML Schema versions and extension prior to the next version of the XFDU Red book.

A set of XFDU XML Schema changes was agreed at the April meeting. These changes were incorporated into the XFDU schema and the XFDU toolkit library as release 1.1. The XFDU schema was frozen until the results of agency prototypes and interoperability testing were available.

New XFDU toolkit library releases were to be made based on improved functionality and bug fixes The most recent version of the XFDU schema and toolkit libraries with upgraded user guides and documentation can be found at the XFDU Project website (http://sindbad.gsfc.nasa.gov/xfdu/). It was agreed that the next formal version of the XFDU Recommendation would be based on the results of testing,

The following sections discuss the major research activities during the period of April, 2005 thru February,2006:

3.1 Specializations of XFDU XML Schema

One of the major reasons for using XML Schema as the underlying notation for the XFDU is the requirement that the XFDU must be able to be extended both to enable new versions of the XFDU Standard to be released over time and to allow third parties (e.g. other standards group, agencies, projects) to create specializations of the XFDU schema that remain compatible with the core schema. These specialization types have many common requirements and are often confused. The definitions of Versioning and Extensibility as used in this document are:

1. Versioning is the process for modifying an XML format over subsequent releases: (e.g. v1.0 -> v1.1 -> v2.0 -> v3.0). Versioning is about evolution, and perhaps extension, of a language, over time.

2. Extensibility is a mechanism that enables new data to show up side by side (or concurrently) with data for a given format. Extensibility is about evolution across space. The Extensions are typically created and maintained by third parties who want to extend the format

During the early phase of XFDU concept definition and schema design a short study of the alternatives and practical considerations of tool support led us to use XML Schema abstract elements and substitutionGroups to define “extension points” where evolution of the XFDU schema was anticipated. A brief concept paper and this evaluation was written in 2002 and several illustrative examples of this technique are included in the current version of the XFDU Specification (5).

During this research period there were two third parties who wanted to specialize the XFDU schema as a basis for their Standard development.

The first specialization activity is called SAFE. SAFE is a standard format for archiving and conveying data within the European Space Agency (ESA) Earth Observation archiving facilities and potentially with the cooperating agencies. The SAFE developers were informed of the XFDU by ESA and became active members of the CCSDS in November, 2004. They had a requirement to produce an operational system in January, 2006.

The second specialization activity is support for the PAIS Draft Standard. The PAIS SIP work, discussed in this document, is intended define a concrete implementation of the abstract SIP defined in the OAIS RM and refined in the PAIMAS (4). The decision to use the XFDU to

develop a concrete implementation of the SIP to aid in the understanding of the PAIS abstract syntax was reached in the April, 2005 CCSDS IPR/DAI meeting.

Both of these efforts wanted to create extensions of the XFDU schema with the expectation that there could be future versions of their standards that would be independent from evolution of the XFDU schema. Both SAFE and PAIS also assumed that users would need to further specialize the schema to better meet community specific requirements.

The SAFE developers were on a very tight time schedule and would have control over all schema specializations in the ESA archive domain. In order to accomplish these goals, XML Schema feature Redefine was used to both specialize the XFDU schema to create a core SAFE schema and to specialize core SAFE schemas to create archive and project specific schemas. The Redefine is known in the XML community as the least understood and inconsistently used feature

The developers of SAFE use XML Schema Redefine to both import and specialize the XFDU XML Schema in a new "SAFE" namespace. Then new XML entities are defined to allow the SAFE XML processor to validate various metadata objects against these standard vocabularies. In order to allow various Earth Science archives/project to further specialize the SAFE standard schemas, SAFE implementers use XML Schema Redefine mechanisms to define Project specific namespaces and schemas. The following paragraphs, which are adapted from the draft SAFE specification, discuss the requirements and approach. The specification also contains the SAFE Schemas and specifically describes the variations from the XFDU.

> During the redefinition or restriction, some features of XFDU are discarded or constrained according to the specific needs of SAFE. SAFE may constrain values of particular attributes, occurrences of elements, and especially add rules of consistence, mechanisms of connection between the various components of a SAFE Product (Manifest file, XML schemas, binary files etc.)
>
> SAFE introduces new types, defining and organizing the product data. These may be complex types such as platformType gathering all data related to the flying acquisition system, or simple types such as platformFamilyNameType defining the platform name.

The SAFE implementers proposed that the use of XML Schema Redefine be recommended as the preferred technique for schema versioning and specialization in the XFDU Recommendation or the XFDU Best Practices Document. This request was rejected as premature. However, a study of recent literature and implementations on XML Schema Namespaces, Specialization and Schema Repository Services Best Practice was approved.

This study is ongoing. The preliminary results of this study are described in the next section.

3.1.1 XML Schema Specialization Best Practices

The initial research into XML Schema Namespaces, Specialization and Schema Repository Best Practices revealed that:

- XML Schema Best Practice documents from major United States Agencies, International Standards Organizations, computer vendors, and CCSDS member agencies were inconsistent and often conflicted in important areas. These results are documented in the Annex B

- Initial investigations of XML Schema Extensions and Versioning mechanisms also revealed that the "seminal papers" had good agreement on the problems in this area. They also agreed on the difficulty of solving these problems without modifications to the XML Schema or the use of specific higher-level protocol constructs such as those in the W3C web services protocols.

A brief study of XML Schema Registry/ Repository products (open source, commercial or government) or publicly available Internet Services revealed no mature products or service offerings that met the requirements for an XML Schema Repository. All the current XML Registry/Repository products and Internet Services were focused on Web Service requirements. The two prominent contenders for the online XML Schema Registries market in the early 2000s, http://xml.org/xml/registry.jsp from OASIS,and biztalk.org from Microsoft, were no longer operational. In our initial study in 2005, indications were that the xml.org website had not been updated since 2003 and the OASIS server update in early 2006 made the pages inaccessible. Microsoft had closed its BizTalk.org registry of XML schemas on July 19, 2002.

During this study period two external efforts occurred that provided validation of the fact that the required XML Schema Versioning and Extension mechanisms were not yet mature. These efforts referenced current activities that could be leveraged to provide a framework that would be interoperable with emerging commercial and government guidance:

- The W3C held a public Workshop on XML Schema Practical Experiences to gather input for the XML Schema 1.1 development. The complete set of input working papers, presentations, session notes and Chairman's final report can be found at http: http://www.w3.org/2005/03/xml-schema-user-cfp.html
- The Federal XML Working Group began the specification of a new version of an XML Schema Naming and Design Rules and Guidance document which covered the same scope as the required XFDU XML Schema Best practices study. This study is ongoing and complete documentation can be found at https://fed-xml-ndr.core.gov/

There still are no complete solutions to these problems. Currently the work on the Universal Business Language Version 2 has revealed that the basic problem of validating Code Lists requires the use of features that are not present in XML Schema and they are investigating Schematron and Namespace-based Validation Dispatching Language (NVDL) as potential solutions. These languages and RELAX NG are parts of ISO/IEC 19757 Document Schema Definition Languages (DSDL), a framework for partitioning XML schema validation problems into several layers and developing focused languages for each layer.

The W3C XML Schema Structure Version 1.1, that should have more features to support versioning, is significantly behind schedule. After an incomplete draft was issued in February 2005, no new versions have been issued.

Conclusions

The status of XML schema versioning and extensibility mechanisms is clearly a concern in the development of the XFDU. However, it appears that waiting for the needed features to be developed in new versions of the W3C XML Schema is a very high risk. Therefore the XFDU research effort should continue the investigation of alternative methods of XML validation. A brief description of a very early phase of this research can be found in section 3.2.4.

3.1.2 Developing XML Schema Specializations for the SIP

The CCSDS Submission Information Package is described in this report. The SIP Standard development team made the decision to base a SIP implementation on the XFDU. The desire of the editors of the SIP specification was to define a set of SIP XML schemas that were formal extensions of the XFDU XML Schemas that would be valid XFDUs but to add SIP identifiers and constraints. There have been several iterations of the SIP XML Schema design between ourselves and the SIP editors which have identified both some strong and some weak areas of XFDU extensibility and mechanisms, an example of using the XFDU validation API for Schematron rules, and some potential XFDU XML Schema changes.

The initial changes requested an XML schema with additional elements and attributes in the contentObject. It had been anticipated that the contentUnit would be an extension point so an abstract element and appropriate substitution Groups had been defined in the XFDU Schema. We merely defined a new concrete implementation of the contentUnit in the SIP namespace with the additional entities

The second change requested was to create global attributes that would appear once and be valid for all objects in the XFDU. While this seemed similar to the previous task, the fact we did not have an extension point built into the XFDU schema for the Information Package Map or the Package Header eliminated the simple solution described above.

We proposed two solutions that could be implemented using the current XFDU Schema

1. Extend the xfdu information package map so the global attributes sipTypeName and sipTemplate can be attached to the information package map . However, since we did not anticipate the information package map as a specialization point there was no abstract class and substitution group available. Therefore, the extension would need to be added to SIPXFDU as the last top level element.

2. Declare the elements as optional on the sipContentUnit. Then, in the instance, put ipTypeName and sipTemplate only at the top level sipContentUnit for the package. Finally, use Schematron to insure that the elements are located once and only once only in the top level sipContentUnit

The PAIS SIP team rejected these two solutions because:

1. The SIP information is located in 2 places. There are 2 package maps that seems like bas schema design would be confusing. (solution 1)
2. The XML schema allows the global information to be repeated. It's up to the Producer to manage the occurrence of this information or else require the use of Schematron (solution 2)

They suggested including the global information in the packageHeader section, as a new complex element. In discussion it was noted that we could create an anyXML element with “lax” validation and have SIP and other third party users specialize the complex element into an element with “strict” validation. However this would create a number of incompatible third party specializations.

This and several other potential solutions, including the use of redefine with a standard initial content unit for SIP, and the definition of an abstract element and substitution groups were discussed by the working group. The SIP designers did not indicate a preference for any of these

solutions and took the matter under advisement.

3.1.3 Observations and Conclusions

The efforts toward SIP specialization of the XFDU XML schema have confirmed the conclusions of the XML Schema Specialization Best Practices Study on the inadequacy of XML Schema specialization mechanisms. It also confirms that the situation becomes much worse if the original schema has not included a specialization point where it is needed

3.2 XFDU Toolkit Library Testing

The NASA XFDU Test effort consists of a series of "performance tests" and "operational tests". The performance tests are designed to identify the XFDU toolkit library APIs that must be optimized to meet real world operational requirements. The operational tests involve the implementation of scenarios using actual data products to provide both experience in the use of XFDUs in actual systems and evidence of functionality and interoperability in the targeted environments. This section summarizes the tests run during the research period, the most recent test results, and preliminary conclusions and issues.. It must be emphasized that the purpose of these tests was to evolve what had been a proof of concept prototype into a reference implementation of the XFDU standard that could be used by "early release" customers. Performance improvements were intended to provide acceptable solutions rather than optimal solutions.

Unless otherwise stated all test were performed using the following:

- IBM ThinkPad T42 with 1.8GHz PentiumM Processor and 1GB of RAM
- Fedora Core 4 Linux OS
- JDK 1.5.0_05

3.2.1 Testing Performance of Underlying Compression and Packaging Utilities

Use Case Description

The ability to extract the manifest from an XFDU Package is important in a heterogeneous, open environment where several third parties have extended the XFDU schema and added new transformation and validation "plug-in's." The goal of this extraction is to use the information in the package header component of the extracted of XFDU manifest to analyze if all the needed mechanisms ("must understand") are present before unpacking the entire XFDU. This set of tests is intended to investigate the performance of common binary archive formats for this operation.

Test Description

The first set of tests investigated extracting a relatively small XML file from a large, compressed binary archive created using the widely available binary archive formats ZIP and JAR. In this set of tests the Linux operating system ZIP command and the java classes provided in JDK under java.util.zip and the java.util.jar packages were used to create the archives and extract the XML file.

Two basic package designs were used in these experiments:

1. 31000 files of various content and size were packaged into a binary archive along with a 715-Byte XML file using a zip command provided with the OS. The resulting ZIP file had size of 639MB.
2. Two DIVx compressed files of average zip 700MB each were ZIPed into a ZIP archive along with a 715-Byte XML file using a zip command provided with the OS. The resulting ZIP file had size of 1.33GB.

The tests were then repeated using JAR as the packaging methodology instead of using the zip command.

Finally the tests were again repeated using tar/gzip as the packaging methodology.

Test Results

Table 1. Package creation and file extraction times versus type of binary file packaging

Test/ Iteration	***Runtime (zip,31000 files)***	***Runtime (jar,31000 files)***	***Runtime (tar/gzip 31000 files)***	***Runtime (zip,2 large files)***	***Runtime (jar, 2 large files)***	***Runtime (tar/gzip, 2 large files)***
Extract XML File / 1	70 ms	68 ms	17.5 sec	32 ms	31 ms	68 sec
Extract XML File / 2-10	.1 ms/run	.1 ms/rum	25 sec	.2 ms/run	.4 ms/run	34 sec
Package creation	428 sec	531 sec	205 sec (using tar with gzip output option) 325 sec (using tar and gzip separately)	204 sec	356 sec	204 sec
Unpacking	243 sec	186 sec	122 sec	128 sec	156 sec	125 sec

Observations

- As a practical matter, it takes an insignificant amount of time to extract a small file from an archive regardless of the number, content and size of files in either a JAR or ZIP archive.
- The fact that, during all the runs, only first iteration took a noticeable amount of time (30 to 70 milliseconds) to extract the file is attributed to the nature of how JVM works. That is, during the first iteration all class loading and initialization takes place. After that the actual extraction time was less than 1 ms for JAR or ZIP
- Package creation times using the operating system ZIP or JDK java.util.jar is 7 to 9 minutes for the 31000 file case and 3 to 6 minutes in the 2 file case. In both cases the unpacking time using JAR or ZIP is 2 to 4 minutes.
- The TAR/GZIP package creation time was approximately 3.5 minutes in both cases while the unpackaging time was approximately 2 minutes. However the time to extract the small file is measured in tens of seconds rather than the almost instantaneous response for JAR and ZIP.

Conclusions and Issues

- No special implementations or best practices are required to allow the XFDU manifest to be extracted prior to unpacking the entire archive from many common binary archive formats(e.g., zip , jar) that contain object indices.
- In the case of binary archive formats like TAR/GZIP that do not have object indices, the manifest would need to be one of the first items on the virtual tape image presented to the application. The effects this would have on the toolkit library and other archive format processors would need to be investigated.

3.2.2 Performance Testing of XFDU Toolkit Library Interfaces

Use Case Description

This use case uses the artificial collections created in the previous use case to measure the performance of the basic XFDU toolkit library APIs and underlying JAVA implementations.

Test Descriptions

1. Create an XFDU package in ZIP format from a directory structure with 31000 files
    a. Using XFDU APIs, a package is created both with and without checksum computation and saved in ZIP format.
    b. Using XFDU APIs, the manifest is extracted from the package.
    c. Using XFDU APIs, a randomly selected dataObject (file) is extracted from the package.
    d. Using XFDU APIs, the package is opened and the files are expanded optionally validating any checksums recorded in the XFDU Manifest object.

2. Create an XFDU package in ZIP format out of a directory structure with 2 DivX compressed files
    e. Steps a-d above

3-4. Repeat tests 1 and 2 using the JAR binary archive format

5-6. Repeat tests 1 and 2 using the TAR/GZIP binary archive format

Initial Test Results

- It was observed that package creation of the 31000 object XFDU using the XFDU API did not complete after 12 hours. Upon closer investigation, it became obvious that the slowness could be attributed to usage of the JAVA implementation of XPath while constructing the Java Object tree.
- As a result, that part of the XFDU API was completely rewritten to perform necessary lookups using only memory object references. This improved performance significantly (from hours to minutes). The observed behavior (in regards to using XPath) only become noticeable when thousands of files are being packaged which results in an XFDU manifest of significant size.

Test Results after Modification to XFDU Toolkit Libraries

Table 2. XFDU Toolkit Library Performance

Operation	***Time (seconds) (zip, 31000 files)***	***Time (seconds) (zip, 2 large files.***	***Time (seconds) (jar ,31000 files)***	***Time (seconds) (jar, 2 large file)s.***	***Time (seconds) targzip, 31000 file)s.***	***Time (seconds) (tar/gzip, 2 large file)s.***
Package creation and saving	Without checksum 882 With checksum 939	Without checksum 995 With checksum 1175	Without checksum 1000 With checksum 1060	Without checksum 960 With checksum 1141	Without checksum 917 With checksum 968	Without checksum 357 With checksum 381
Writing of files (copying of bytes to zip stream)	*819*	*990*	*940*	*955*	*865*	*354*
Package size	*≈850 MB*	*≈1.3 GB*	*≈850 MB*	*≈1.3 GB*	*≈720 MB*	*≈1.4 GB*
Manifest size	*≈11MB*	*≈1.3KB*	*≈11MB*	*≈1.3KB*	*≈11MB*	*≈1.3KB*
Manifest extraction	25	.023	23	.020	70 (45 seeking +25 extracting)	65 (65 seeking +.02 extracting)
File extraction	Depends on file size	169 seconds	Depends on file size	180 seconds	Depends on file size + tar position	File 1: 68 sec File 2:102 sec (33 seeking +69 extracting)
Package opening including validation	Without checksum 400 With checksum 422	Without checksum 175 With checksum 195	Without checksum 372 With checksum 398	Without checksum 183 With checksum 193	Without checksum 185 With checksum 190	Without checksum 139 With checksum 148

Other Tests

Deletion of a dataObject and all references to it from a manifest containing 31000 data objects and totaling 11MB in size:

- with Commons JXpath - 26000 milliseconds
- with binary search and pure in-memory operations - 8 milliseconds

Observations

- The time required for packaging and unpacking the test XFDUs using the XFDU Toolkit interface was two-three times longer than the time needed to create packages in the previous test of the native interfaces. However the major reason appears to be the performance of the JDK implementations of the compression methods rather than added processing needed to create XFDU structures.
- The TAR/GZIP tests confirm the conclusions in the previous section that the extraction of the manifest and extracting specific files will not work unless the underlying package has indices to enable efficient random access and extraction of single objects
- The delete object test appears to confirm degradation of performance while using Xpath on a large XML tree (or Java Object graph representing such a tree) shows general problem with doing Xpath queries on large XML tree.

3.2.3 Usability of XFDU Information Model and Toolkit Performance with Current Archived Data Products

This section describes testing that involved the XFDU packaging of NASA Planetary Science data and USGS agricultural data.

3.2.3.1 Transfer of PDS Archive Volume from JPL to NSSDC

Use Case Description

The National Space Science Data Center (NSSDC) is currently using a Standard Formatted Data Unit (SFDU) based data packaging approach to wrap data from producers to generate a SIP and also to form the AIP. The resulting package contains an NSSDC attribute object and the individual data files. The NSSDC attribute object contains all the attributes about the data files that NSSDC desires for its internal management. NSSDC has been working with the Planetary Data System (PDS) at the Jet Propulsion Laboratory (JPL) to package their large volume directory structures that eventually will be over 20 GB. The purpose of this use case is to use the XFDU to generate a similar package and observe any issues.

A PDS data volume and an NSSDC Attribute Object corresponding to that data volume were provided for this testing. The PDS data volume consisted of a standard PDS data volume directory structure and was 600MB in size. The NSSDC attribute object was 439 KB in size and was taken from an NSSDC packaging of the PDS volume using an NSSDC AIP implementation conforming to the SFDU standard. The use of the pre-existing NSSDC attribute object simplified the gathering of attributes values for incorporation into the XFDU.

Test Summary

The following software was developed for this test:

1. A simple Parameter-Value-Language (PVL) parser was written to parse the NSSDC attribute object whose attributes were expressed in PVL.
2. A directory crawler and package creator was written to automate the task of crawling the directory structure, identifying the appropriate attributes from the NSSDC attribute object, and calling the XFDU library to create the XFDU package.

The following steps were involved in the actual test:

1. The above-specified software would crawl the PDS data volume directory structure obtaining information about each file and directory, and it would use that information for XFDU content unit creation. For each file found, the appropriate meta-information was extracted from the NSSDC attribute object and saved into a new directory structure created for metadata. This structure paralleled the PDS volume directory but was one level down from the root.

2. Directory path and file name information from the NSSDC attribute object was used for creation of directories and files. The checksum on each data file from the NSSDC attribute object was was extracted and included in the XFDU manifest dataObject description for the file within the byteStream element. A new checksum on the data file was calculated and included in the XFDU manifest dataObject at the dataObject element level. They were found to agree.

3. Also in the XFDU manifest, each file from the PDS volume was correlated to the appropriate file with metadata from the NSSDC attribute object via creation of an appropriate content unit. Content units for directories were created to contain the Content Units of the data files and other directories. In a separate test, a manifest version was also made where all Content Units contained only data files and there was no nesting of Content Units.

4. An XFDU package in the form of ZIP file was created. The ZIP file included the PDS volume directory tree, the NSSDC metadata directory tree mimicking the PDS volume tree but shifted by one level, and the XFDU manifest file describing the packaging of the PDS data volume.

5. The package was unzipped using the XFDU Library. Each file's integrity (checksum) was verified to make sure that all files inside were intact.

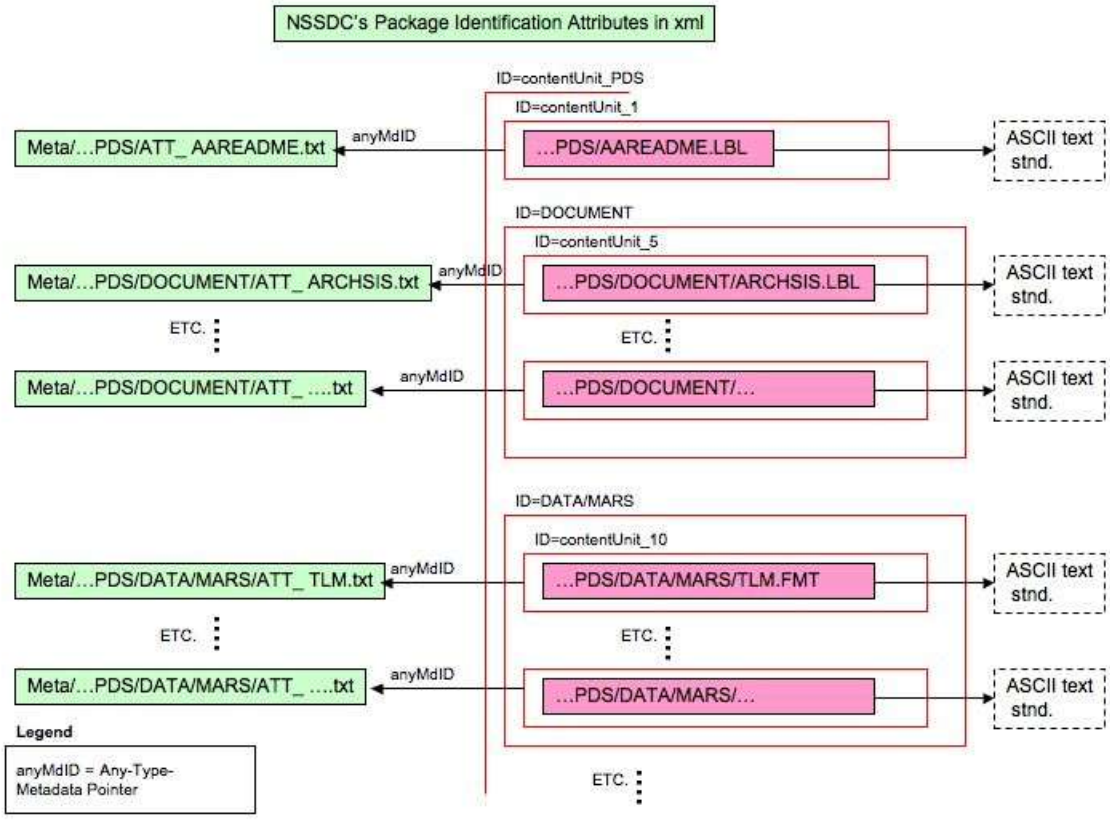

Figure 5. Logical view of PDS data files and NSSDC metadata objects in XFDU package

The logical structure of the resulting XFDU is shown in figure 5. The original PDS supplied data files are shown in 'red' and are contained in Content Units. They are also associated with Representation Information via mime types. No attempt was made to improve this description of the Representation Information, however such information is available from the PDS Standards Reference at http://pds.jpl.nasa.gov/documents/sr/index.html.

Each Content Unit, holding a data file, also identified the associated metadata file via use of the AnyMdID attribute. These metadata data files are shown in 'green'. In addition, some general

attributes about the package from the SFDU were incorporated into the XFDU as an xml 'package identification' group of elements.

Test Results and Observations

- It took 8-12 person-hours between the PDS Systems Programmer and the XFDU Library developer to code the test
- According to the PDS System Programmer, who was not intimately familiar with the XFDU API, it was relatively easy to use
- It took, on average (based on 3 consecutive runs), 4.5 minutes to create the ZIP package (including metadata extraction from the NSSDC attribute object). This included 600MB of PDS data and metadata from the NSSDC attribute object
- The resulting ZIP package was 450 MB in size
- Unzipping of the package took on average 3 minutes (based on 3 consecutive runs)
- The use of nested content units did not appear to make any noticeable difference in package/manifest creation time when compared to runs with all content units at the same level.
- While the NSSDC attribute object involved did not exhibit the most complex view of such an object, there were no major issues in mapping the attributes to the XFDU. One minor issue is that the NSSDC SFDU for this data includes a checksum over the NSSDC attribute object itself. Currently there is no standard provision for a checksum over the XFDU manifest file.

Issues and Conclusions

The XFDU was able to package a PDS data volume of 600 MB containing many levels of directories, and was able to return that directory volume with verification via checksum validation. The packaging retained all of the metadata and its associations as recorded in the SFDU packaging.

The packaging and unpackaging performance, using the created software and XFDU library was quite reasonable especially given that no optimization has been done to the library.

The lack of overhead for nesting contentUnits and the difficulty of creating consistent "Order" attributes indicates that nesting should be the ruling structuring indicator and the order attribute should simply be removed or considered a text attribute to assist human understandability

The PDS Systems Engineer was concerned about future data products with data volumes in excess of 20 GB. The XFDU developers discussed an XFDU schema enhancement to support Logical Volumes mapped across multiple XFDU packages. This type shown below was initially proposed in response to a PAIS SIP requirement. The PDS Systems Engineer felt this proposal would solve his issue.

```
<xsd:complexType xmlns:xsd=http://www.w3.org/2001/XMLSchema
name="sequenceInformationType">
                    <xsd:attribute name="sequenceNumber" type="xsd:long" default="0"/>
                    <xsd:attribute name="numberInSequence" type="xsd:long" default="0"/>

</xsd:complexType>
```

There was broad agreement for further testing involving transformations, relationships and much larger data volume. A paper about the extended testbed was proposed for SpaceOps 2006 and an abstract was submitted, This paper titled "A Distributed Testbed for the Exchange of XML Aggregated Data Exchange Products for Mission Operations" (7) was accepted and will be presented in June, 2006 and included in the Proceedings.

3.2.3.2 Developing an XFDU Version of USGS Data Submitted to NARA during ERA Prototyping

Use Case Description

This test involves the use of USGS data, obtained by NARA as test electronic records, to create a package using the XFDU draft standard. NARA traditionally organizes its records into Record Group, Record Series, File Units, and Items, with mandatory and optional attributes for each being made available for searching and management.

This test begins to flesh out the relationship of typical NARA record documentation to the capabilities of the general, and flexible, XFDU specification. It goes beyond this to take advantage of some XFDU relationship capabilities supporting OAIS concepts designed to lead to fully understandable, readily transferable, and preservable information objects. In the process, a better understanding of some strengths and weaknesses of the typical NARA record documentation approach to electronic records should emerge. At the same time, strengths and weaknesses of the XFDU packaging and description capabilities should also emerge.

Test Summary

The USGS data selected consists of 6 sets of agriculturally related information known as ag_chem, ag_stock, ag_land, ag_expn, ag_crop1, ag_crop2. While all these data are packaged into a single 38 MB zip file for this test, the primary concentration is on an analysis of the ag-chem data packaging. The other data are similar in nature and do not add significantly to the exploration of relationships.

The ag_chem data that we have chosen to package consists of 1987 statistics on agricultural chemical use across the US. These statistical results are provided in two choices of format – one called SDTS and the other called arcINFO. Accompanying these data are two additional files, called documentation files, with one giving the documentation written using XML and also conforming to the FGDC metadata standard. The other is a map image showing the percent of land in each US county that received insecticide during 1987. Percent of land covered by insecticide is just one of a large number of attributes available from the ag_chem data.

Future users of the ag_chem data will need to understand how to access and interpret the digital files being provided. Therefore additional information describing structure and meaning of the data files must also be preserved. Specifically this is the format specification for the SDTS files, for the arcINFO file, for the FGDC documentation file, and for the image file. We have taken it upon ourselves to act the part of the archive by either identifying common standards used, or by actually acquiring standards from a Web search so that they can be incorporated into the package. We have directly incorporated the standard specifications for SDTS, arcINFO, and FGDC metadata into the package. However if the archive already has these standards in an accessible location, the package could simply link to them via standard xml-based reference capabilities. Regardless, the result is a package that can be guaranteed to be more understandable over time than a similar package without identification and access to the format standards involved.

We have also incorporated text descriptions giving additional context and some provenance information for the SDTS data, the arcINFO data, the documentation files, and the corresponding standards specifications Finally, we obtained from NARA descriptive metadata for ag_chem at the Record Group, Series, and File Unit levels. We're led to understand that item level descriptions are not generally produced, at least for electronic records. We've incorporated the descriptive metadata into the package and linked it to the appropriate level of aggregation through identifiers specifically denoting 'descriptive metadata.' The resulting package can be viewed as a type of Archival Information Package in the OAIS sense. It might also be viewed as a type of Submission Information Package prepared by a data producer. In this case, it is assumed that the producer has negotiated with the archive and has been persuaded to include the additional metadata. The producer would have worked with the archive to define and create such metadata ahead of the actual submissions. Note that no effort has been made to incorporate any attributes reflecting the draft PAIS SIP standard because this work is not currently sufficiently mature.

Test Results and Observations

A schematic of the XFDU package showing the contained data objects and their logical relationships is given in figure 6. Additional metadata, including other pointers, used to implement these relationships are not shown. The full xml manifest file, and the full zip package, can be obtained from http://sindbad.gsfc.nasa.gov/xfdu/shared/ag4_xfdu.zip.

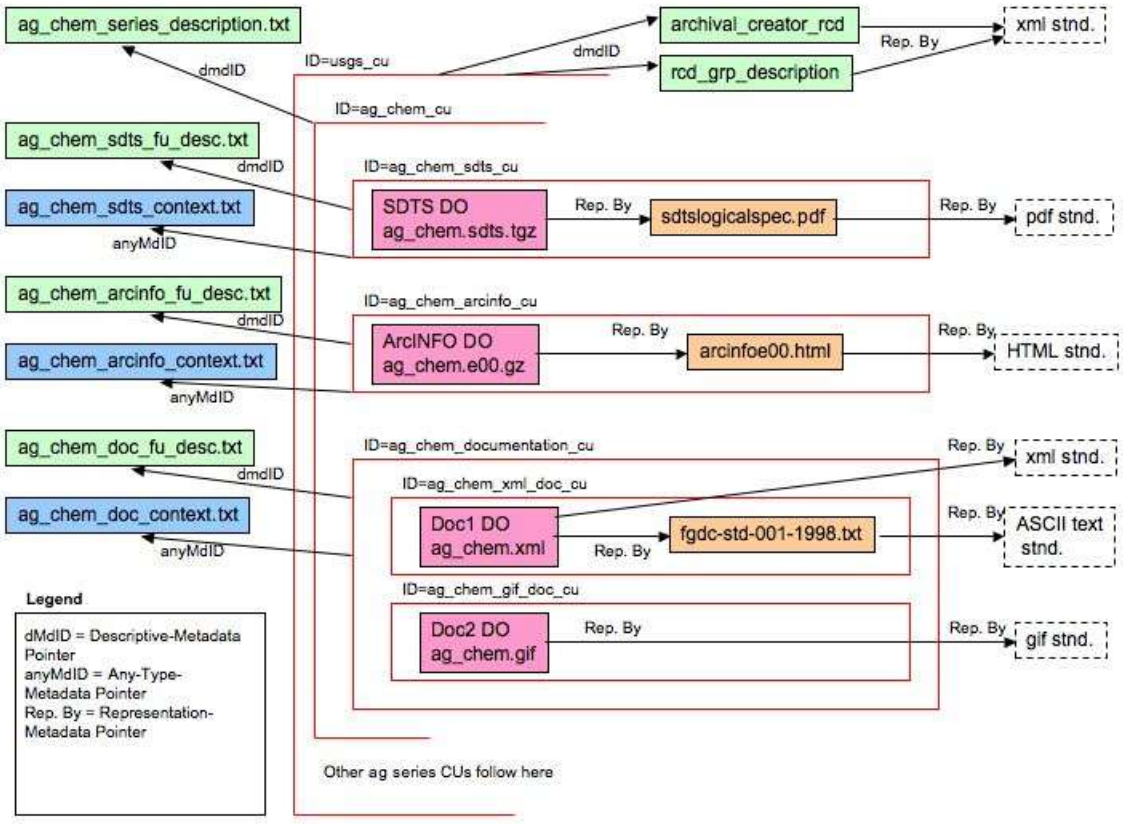

Figure 6: Schematic of the NARA/USGS XFDU showing data objects and their logical relationships

The data objects involved are shown as colored rectangles. Data objects that are the primary target of preservation, as obtained from NARA test data, are in purple. Their associated format descriptions, which were collected from the Web by us for inclusion, are shown in orange. Additional descriptions that we prepared, addressing context and provenance information, are shown in blue. Typical NARA descriptive metadata, obtained by request from NARA, are shown in green. Arrows between them show relationships whose type is given by their labels. For example, an arrow with a label of "Rep. By" denotes that the object pointed to is the Representation Information for the base object. Transparent rectangles, or partial rectangles, with red lines denote XFDU content units. Dotted rectangles are used for data objects not present in the package and stand for various format standards such as XML, PDF, etc. They are expressed as mime types in this package. An archive will want to ensure that descriptions of these formats are also preserved and available. Links to such descriptions could then also be included in the package.

As can be seen from Figure 5, a highest-level content unit (CU) is associated with the record group and is used to logically contain CUs for each record series. At this level, an archival creator record (green) and a record group description (green) are associated as descriptive metadata (dmdID). They have been transformed, from the text that NARA provided, into XML data and incorporated directly into the XFDU manifest file.

At the next level down are the series CUs, and only the ag_chem series is expanded in the figure. The other series in the package look similar. The ag_chem series CU has a single link leading to NARA provided descriptive metadata (green), which we've put into a file called 'ag_chem_series_description.txt'. It could have been transformed into xml and put directly into the manifest, but we decided to keep it, and the other NARA provided descriptions, as separate text files that can be found in the zip structure. The ag_chem CU contains three other CUs as the same level.

The first ag_chem CU, with ID=ag_chem_sdts_cu, is a NARA file unit and is holding the SDTS data object(purple) which is a tar-gzip file that, when expanded, conforms to the SDTS standard. This is shown by the 'Rep. By' link to the SDTS logical specification held as a PDF file (orange). Also linked to this CU is a 'context' description(blue) using the 'anyMdID' attribute. This description talks about the relationships between the SDTS data object and the other primary data objects, and it gives some provenance information. Also linked to this CU is the NARA provided file unit descriptive metadata(green) called 'ag_chem_sdts_fu_desc.txt'.

The second ag_chem CU, with ID=ag_chem_arcinfo_du, is a NARa file unit and is holding the ArcINFO data object (purple) which is a gzip file that, when uncompressed, conforms to the arcINFO standard. Its associated metadata have the same relationships as described for the SDTS CU.

The third ag_chem CU, with ID-ag_chem_documentation_cu, is a NARA file unit and is holding two other CUs. It has a link to a 'context ' description (blue) that describes relationships to the SDTS and ArcINFO data objects. It also has a link to the NARA provided file unit descriptive metadata (green) called ag_chem_doc_fu_desc.txt. The first contained CU, with ID=ag_chem_xml_doc_cu, holds documentation describing the data in the SDTS and arcINFO data objects. It is in an xml form and is further described by the FGDC metadata standard (orange) in the file 'fgdc-std-001-1998.txt'. The second contained CU, with ID=ag_chem_doc_cu, holds an insecticide coverage image in the gif format.

Conclusions and Issues

For the USGS Ag series of data records, there was no problem providing a view of the NARA hierarchy of record aggregations and the association of NARA descriptive metadata with the appropriate aggregation level. However we note that the series level description provided by NARA not only addressed the file units to be aggregated, but it broke out descriptions down to the level of identifying the total number of files when the sdts tar-gzip and arcinfo gzip files were uncompressed and expanded. We would have thought that the series level descriptive metadata might have been limited to talking about the contained file units only.

We were also able to associate format descriptions with the 3 primary data objects whose formats are not widely used, thereby resulting in a package of information that has long lived potential. In order to address relationships among the CUs, including addressing the history of how the components of each were obtained, we used a 'context' metadata description for each of the three file unit CUs. We also could have included an attribute on the SDTS CU and on the ArcINFO CU that linked to the ag_chem.xml documentation data object. This relationship would have been 'anyMD', or just 'this is related metadata'. Formal relationships among CUs are not currently supported but is a likely future topic for XFDU evolution.

What is not visible from the logical view is that most metadata links are implemented by pointers to a metadata object, then the metadata object points to a data object, and then the data object points to the actual byte stream located elsewhere in the zip structure. In addition, the SDTS and ArcINFO data objects have transformation information associated with them that allows software to uncompress and/or unpack (i.e., untar) them. Generating this test package also pointed out that the draft XFDU standard does not make clear what to do when one transformation results in multiple objects (e.g., untar) and then a subsequent transformation (e.g., decrypt) is to be applied. This will be addressed to the XFDU developers.

3.2.4 Implementing Advanced XFDU Functionality in the XFDU Toolkit Library

The study of XML Schema specialization mechanisms indicates the importance of alternative validation mechanisms to enable the level of specialization required for effective use of XFDU as base schema for reuse in defining more specific packages for domain interoperability.
Currently the XFDU incorporates Schematron rules for the optional validation of semantic rules specified in the XFDU recommendation. The experience with the PAIS SIP schema development indicated that while this technique could be used to validate structural Specializations, the complex rule definition and the apparent inconsistency of the XML schema would be significant issues. The following experiment is an effort to allow verification of individual XML formatted data and metadata using included or referenced XML schema without involving the specialization of the underlying XFDU schema.

3.2.4.1 Validation of Data Objects using XSD Descriptions

The XFDU Package created during testing with JPL was used as basis for this test. The package was chosen since it was already available and contains a reasonable number of data objects and associated PVL objects. To conduct the test one of the PVL attribute objects existing in the JPL XFDU package was converted to XML. An XML schema governs the XML that was generated. The file reference of each of 296 data objects in the package was replaced with a reference to the created XML file. Each data object's repID attribute was filled with the reference to the metadata object that encapsulated the reference to the created XML schema. Finally, necessary adjustments were made to the informationPackageMap to include content unit pointing to the XML schema metadata.

The object of the test was to use created XML schema to dynamically validate each of the XML-formatted data objects. For this purpose a validation handler was created that implemented the ValidationHandler interface defined in the NASA XFDU Java library. This handler was used as custom validation handler plug-in for the validation API. The plug-in is then used during execution of XFDU API for validation of an XFDU package. The validation handler used Sun's Multi Schema Validator in combination with SAX Parser to perform actual validation. For the purposes of the test intentional errors were introduced in the sample XML file. During the validation of each data object found in the package, the validation handler would traverse to its representation metadata via the value of repID attribute and retrieve the corresponding schema. Then, the validator would validate the content of the data object against the schema. Validation errors were collected during the execution. For example, the test XML was modified to include a string value in an element that requires an integer value thereby triggering a validation error.

Test Results and Observations

To measure performance overhead introduced by validation, the test was run 10 times. Validation of 296 identical data objects of 1.8 KB each against one XML schema resulted in average overhead of 1.8 seconds. This is an average of 5 milliseconds/object. The errors introduced in the manifest were reported as: "ERROR: com.sun.msv.verifier.ValidityViolation: "error value" does not satisfy the "integer" type ".

An additional scalability test indicated that performance of this technique would increase linearly for up to 5000 objects

Conclusions

- The validation handler used Sun's multi schema validator in combination with a SAX Parser to perform actual validation of XML-formatted data objects via associated representation metadata in the form of their XML Schema (or DTD) doesn't introduce any significant overhead of XFDU processing.
- The overhead would vary depending on parameters such as the size of data objects, number of schema's, size of each schema and number of errors in each data object.
- The XFDU library validation API can be used to plug in such validation tools.
- This technique would enable projects to validate more specialized XFDU instances without creating complex and potentially conflicting specification of the underlying XFDU schema.
- It is important to determine the scalability of this and other non-XML schema validation techniques

3.2.4.2 Scalability Testing

Use Case

The mechanism of validating individual data objects provides significant added value to the XFDU Toolkit Library and may provide a practical solution to the unresolved issue of extending the XFDU XML Schema by third parties. To validate the utility of this technique, the scalability to a realistic maximum of the various factors that effect performance is needed. The following was an informal scalability test using the number of data objects as the independent variable.

We also created a scalability test for Schematron using the size of the manifest as the independent variable. This test was suggested by some experience during the extension of the XFDU Schema as a proposal for SIP.

Test Results and Observation

Validation via XSD with 5 schemas and 5 test XML files spread over the sample:

Table 3. Validation times using XSD

Number of Data Objects	Total Time for Validation	Validation time/object
296 data object	1.5 seconds	5 ms
1200 data objects	4.5 seconds	3.8 ms
4888 data objects	14.8 seconds	3.0 ms

Schematron validation

Table 4. Validation Times using Schematron

Manifest Size (MB)	Total Time for Schematron validation	Time/Manifest MB
1.5 MB	133 seconds (2 min)	88.67 sec/MB
5 MB	1437 seconds (23.9 min)	287.4 sec/MB
11 MB	13800 seconds (230 min)	1254.5 sec/MB

Issues and Conclusions

The validation of individual data objects appears to be very scalable through 5000 objects. The observation that the time/object decreases as the number of objects increase is probably due to a fixed or slowly increasing initial overhead that is being distributed over an increasing number of objects.

The Schematron mechanism does not seem to scale very well. Given that Schematron uses XSLT and XPATH, the problem could be similar to the JXPATH issue we encountered in the XFDU creation tests. However, we didn't implement Schematron so it is doubtful we can optimize it. We will need to include manifest size limits for the use of Schematron to validate XFDU rules.

Clearly, many more scalability test are needed however these tests indicate that using XSD validation on individual data objects appears to be a viable mechanism for allowing third party extensions that are independent of other extensions.

4 Issues and Decisions Regarding PAIS Standardization

The PAIS standard (also referred to as the SIP standard) is being edited and led by colleagues in France at the Centre National D'Etudes Spatiales (CNES). Our research work during this period has been to critique the concepts and the document, and to perform some initial testing of the concepts. This work is discussed under the three categories given below.

4.1 Generality and Understandability of the Concepts

If a standard is to be successfully implemented, it must be as clear and as unambiguous as possible to those expected to implement and use the standard. A major critique of the draft document under review in May 2005, and discussed at the May CCSDS workshop in Athens, Greece, was the difficulty in following the various concepts and their inter-relationships.

It should be noted that the subject matter is admittedly complex as one is attempting to provide a general approach along with standard mechanisms that can be specialized as needed to support formal modeling of data objects being submitted to an archive. The initial approach was to define a generic model for a Descriptor which could be specialized by a domain, further specialized for an Archive Project, and then instantiated to be the "Model of Transfer". There was also the concept of a generic packing slip, or 'Generic SLIP Model' that would be specialized for an Archive Project and included with the "Model of Transfer". These models would be exchanged between Producer and Archive to arrive at a common understanding of exactly what was to be transferred. It was also proposed that actual SIPs would be formed by instantiation of the SLIPs along with the actual data objects in some type of transfer container.

As there was no recognition of any aspects about the underlying transfer containers, this generality made it especially difficult to follow the concepts and to conceive of standardized implementations. The concept of SLIPs was dropped in favor of looking at a direct mapping to the XFDU container and its supporting services. This has lead to the SIP model as discussed above and the intent to provide a mapping to XFDU in the SIP standard.

A further complication was the specification of 4 different Descriptor models corresponding to the DOs, CDOs, Collections of DOs, and Collection of CDOs. There is now just one basic Descriptor model to cover all these types of descriptor usages.

4.2 Organization of the Document

Despite the simplification addressed in Section 3.2.1, the overall organization of the document has changed very little. It is still difficult to follow and to know just what is to be standardized. Our analysis is that it provides a lot of process and work flow, and activities that might be done as described or in some other way, along with some abstract and concrete syntax for Descriptors, both data object and SIP. Work to be done is to clearly identify what is to be standardized and to present this following a short section on overall context. The issue of specialization by domains, archives, and Archive Projects can be dealt with in an Annex. This should make it much easier to understand and to apply to various test cases.

4.3 Need to Support Differing Data Producer Views as to their Materials for Submission

The current data model underlying the standard is that of 'Descriptors', or sets of attributes about some type of 'data object.' While a hierarchical view of data objects makes good sense, it is not clear that the concepts of DOs, CDOs, and Collections of these, with a one-way relationship from CDO to DO, are sufficiently general or useful even for a limited space agency domain. In order to get Producers to cooperate in generating a formal model of what is to be transferred, there must be a framework in which to do this that is easy for them to adopt and modify. It must also work for the Archive. Given the wide variety of Producers, even within the Space agencies, we have suggested that work needs to be done to create test frameworks that can be presented to Producers

to gauge their effort in cooperating in the modeling process. The SIP standard needs to encourage Producer cooperation in every way possible. In our view, this is the most critical and difficult challenge to arriving at a useful SIP standard.

Producers will often have very different understandings about the types of data objects they control, how they see the relationships among them, the extent to which the data objects have documentation that can be used to understand them, the role of supporting software, and even the importance of processing history. A given archive will have its own view on these matters. Somehow, these views need to be bridged during any given Archive Project and the SIP standard needs to aid or be neutral toward, but not impede, the bridging process.
The Archive and the Producer need to establish a common vocabulary and a set of common concepts associated with the modeling of the data to be submitted. The current draft SIP document argues that Producers should not have to understand the OAIS reference model, which is certainly true. However we do not believe this means that some OAIS data modeling concepts are not appropriate for establishing a common understanding between Producer and Archive. To explore this further, we generated a paper entitled 'Toward a Producer Questionnaire to Facilitate Formal Modeling of Archival Submissions' (8) that addresses the presentation of data modeling concepts that might be adopted by Producers who have no knowledge of OAIS or Archive specific terminology. Although the paper was not complete, as it was lacking some test cases, it was provided to the most recent CCSDS workshop held in Toulouse, December 5-8, 2005. Further work is needed to clarify the objectives and approach, and to try it out on potential Producers. Results may impact the DO, CDO view and require a more flexible identification of Descriptor types and their relationships

5 Summary Status, Findings, and Known Issues

5.1 Status of Specification Efforts

5.1.1 XFDU Status

The XFDU Structure and Construction Rules is currently being updated for release as a CCSDS Redbook for agency review. All schema changes that were approved in the December 2005 CCSDS IPR Workshop are being incorporated and the explanatory text and examples are being revised based on the schema changes and comments made by members of the IPR Working Group on the current version of the Specification. The draft version of this material and an update to the XFDU toolkit library incorporating both the schema updates and the additional validation and transformation capabilities discussed in this report should be available at the end of March.
We will also be writing an tutorial and Best Practices Guide that will document our lessons learned in the areas of XFDU design decisions for various sizes and designs of data products, an XFDU toolkit library AIP usage guide, a guide to XFDU specialization and a tutorial on the use of Transformation, packaging and validation plug-ins. The first version of this document should be available at the beginning of May and will accompany the XFDU Structure and Construction Rules Recommendation for agency/public review.

5.1.2 PAIS Status

The draft PAIS standard (6) is currently a CCSDS White Book which means that is still being developed by the working group and is not yet ready for a formal review by the participating agencies. There are incomplete sections, and it is questionable that all of the sections will remain in the actual standard. The concepts and the ability to model data are being tested by the agencies

through the use of test cases. These results will be conveyed to our CNES colleagues and the working group, and will be discussed at the next standards meeting currently scheduled for Rome, June 12-16, 2006.

5.2 Findings and Issues Summary

5.2.1 XFDU Findings

There are 4 major findings:

1. The status of XML schema versioning and extensibility mechanisms is clearly a concern in the development of the XFDU and in the use of XML, described by XML Schema, as an Archival Format. This finding was a conclusion of the XML Schema Specialization Best Practices Study and was confirmed by the practical efforts to extend the XFDU XML schema to create the SIP mechanisms.
2. The mechanism of validating individual data objects using the Sun Multi-schema Validator throuogh the XFDU toolkit validation interface provides significant added value to the XFDU Toolkit Library and may provide a practical solution to the unresolved issue of extending the XFDU XML Schema by third parties. Preliminary testing of the current implementation has shown this mechanism to be highly scalable.
3. The approach of developing one or more high quality reference implementations in parallel to the specification has proven very valuable in identifying unclear portions of the specification and building confidence of potential users. This was demonstrated by the NSSDC/PDS test-bed experience and commitment to expanding the scope of the test-bed.
4. The modeling of the USGS/NARA products confirmed the flexibility of the XFDU information model and the ability to easily express OAIS Information Model Representation Networks

5.2.2 XFDU Issues

The current research effort has provided a good platform for further research but several important areas require further investigation. These issues include:

1. Definition of mechanisms to enable effective implementation of behaviors and compact definition of relationships for XFDU version 2. This was anticipated due to resource availability in the current research period.
2. There is a requirement for much more performance and scalability testing using anticipated data loads for NARA ERA As a prerequisite for this effort the XFDU Toolkit library will need to be migrated to a 64 bit environment
3. There needs to be much more testing of the use XFDU structures using real existing data
4. There needs to be a study on the interoperation or at least the co-existence of the XFDU and METS and best practices document to assist in the selection
5. The study of XML Specialization and Validation should be extended an include the definition of an Archivable Profile of XML Schema and the potential use of of ISO/IEC 19757 Document Schema Definition Languages (DSDL) languages as an alternative to XML schema

5.2.3 PAIS Findings

There are 2 major findings:

1. The use of Descriptors, as a set of attributes focused on a group of one or a few files, to construct the formal model for negotiation between the Producer and the Archive, appears promising despite issues with the current Descriptor modeling.

2. A PAIS standard that addresses the use of Descriptors and that maps the formal model view into the XFDU standard's capabilities still appears promising.

5.2.4 PAIS Issues

There are 3 major issues:

1. The current draft (6) is too difficult to understand and is also ambiguous in a number of places. We are working to arrive at an agreed reduce scope and an unambiguous presentation.
2. The current Descriptor model, although somewhat ambiguous, appears to be overly constrained and thus insufficiently general to be readily applied to a wide variety of data as understood by Producers.
3. The SIP model is incomplete and needs to be mapped to the XFDU capabilities. However this can't be fully completed until the Descriptor model is fully agreed.

References

1 *Reference Model for an Open Archival Information System (OAIS)* . Recommendation for Space Data System Standards, CCSDS 650.0-B-1, Blue Book, Issue 1, Washington, D.C.: CCSDS, January 2002 [Equivalent to ISO 14721:2002].

2. *Standard Formatted Data Units - Structure and Construction Rules*. Recommendation for Space Data System Standards, CCSDS 620.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, May 1992. [Equivalent to ISO 12175]

3. "*Report on XML Packaging*," World Wide Web Consortium, http://www.w3.org/1999/07/xml-pkg234/Overview, July 2000.

4. *Producer-Archive Interface Methodology Abstract Standard*. Recommendation for Space Data System Standards, CCSDS 713.0-B-1, Blue Book, Issue 1, Washington, D.C.: CCSDS, May 2003 [Equivalent to ISO 20652].

5. XML Formatted Data Unit (XFDU) Structure and Construction Rules. Draft Recommendation for Space Data System Standards, CCSDS, White Book, Washington, D.C.: CCSDS, September 15, 2004. (http://sindbad.gsfc.nasa.gov/xfdu/pdfdocs/iprwbv2a.pdf)

6. *Producer-Archive Interface Specification*. Draft Recommendation for Space Data System Standards, CCSDS 651.1-W-03, White book, Issue 3, Washington, D.C.: CCSDS, December 2005. (http://public.ccsds.org/sites/cwe/moims-dai/Public/Draft%20Documents/paimas_implementationWB_02.doc)

7. *A Distributed Testbed for the Exchange of XML Aggregated Data Exchange Products for Mission Operations*, L. Reich, et al., to be published in SpaceOps 2006 Proceedings, Rome, Italy, June, 2006.

8. *Toward a Producer Questionnaire to Facilitate Formal Modeling of Archival Submissions*,

edited by Sawyer, D. M., provided to CCSDS workshop in Toulouse, France, November 2005. (http://sindbad.gsfc.nasa.gov/xfdu/pdfdocs/questions_for_producers_3.pdf)

Annex A: XFDU Validation Schema and Instance

The following demonstrates the sample XML that was used in the XML Validation test:

```
<STREAM_STRUCTURE>

<STREAM_INSTANCE_POINTER>6</STREAM_INSTANCE_POINTER>
      <DIRECTORY_PATHNAME>./CATALOG/</DIRECTORY_PATHNAME>
      <ORIGINAL_STREAM_STRUCTURE>
        <STREAM_TYPE>7-BIT ASCII</STREAM_TYPE>
        <STREAM_TYPE_TO_PACKAGER>BINARY</STREAM_TYPE_TO_PACKAGER>
        <ORIGINATING_SYSTEM>Linux</ORIGINATING_SYSTEM>
        <DATE_TIME_CREATED>2002-01-07T20:27:58</DATE_TIME_CREATED>
        <DATE_TIME_LAST_MODIFIED>2002-01-
07T20:27:58</DATE_TIME_LAST_MODIFIED>
        <FILE_ORGANIZATION>sequential</FILE_ORGANIZATION>
        <RECORD_FORMAT>undefined</RECORD_FORMAT>
        <RECORD_CONTROL>none</RECORD_CONTROL>
        <STREAM_SIZE_BYTES>1453</STREAM_SIZE_BYTES>

<MAXIMUM_RECORD_LENGTH_BYTES>0</MAXIMUM_RECORD_LENGTH_BYTES>
        <FILE_NAME>CATINFO.TXT</FILE_NAME>
        <CRC_TYPE>NSSDC_A:V0</CRC_TYPE>
        <CRC>9d4bcd87</CRC>
      </ORIGINAL_STREAM_STRUCTURE>
      <CANONICAL_STREAM_STRUCTURE>
        <STREAM_TYPE>BINARY</STREAM_TYPE>
        <STREAM_RECORD_DELIMITER>NONE</STREAM_RECORD_DELIMITER>
        <STREAM_SIZE_BYTES>1453</STREAM_SIZE_BYTES>

<MAXIMUM_RECORD_LENGTH_BYTES>0</MAXIMUM_RECORD_LENGTH_BYTES>
        <CRC_TYPE>NSSDC_A:V0</CRC_TYPE>
        <CRC>9d4bcd87</CRC>

<RECOMMENDED_FILE_NAME>CATINFO.TXT</RECOMMENDED_FILE_NAME>
        <PROCESSING_REPORT>FsGET_FN-P_UNA PASS: found ASCII with no carriage
control undefined records, AIPGEN-W_BA WARN:  expected BINARY, but found only
ASCII</PROCESSING_REPORT>
        <FORMAT_IDENTIFIER>ZDEFAULT</FORMAT_IDENTIFIER>
        <ORDERED_APPLIED_ENCODINGS>none</ORDERED_APPLIED_ENCODINGS>
        <ID_OF_ENCODED_FORMAT>ZDEFAULT</ID_OF_ENCODED_FORMAT>
      </CANONICAL_STREAM_STRUCTURE>
      <SUPPORTING_ATTRIBUTES/>
</STREAM_STRUCTURE>
```

The following demonstrates the schema governing the previous XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
 <xs:element name="STREAM_STRUCTURE">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="STREAM_INSTANCE_POINTER"/>
    <xs:element ref="DIRECTORY_PATHNAME"/>
    <xs:element ref="ORIGINAL_STREAM_STRUCTURE"/>
    <xs:element ref="CANONICAL_STREAM_STRUCTURE"/>
    <xs:element ref="SUPPORTING_ATTRIBUTES"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="STREAM_INSTANCE_POINTER" type="xs:integer"/>
 <xs:element name="DIRECTORY_PATHNAME" type="xs:string"/>
 <xs:element name="ORIGINAL_STREAM_STRUCTURE">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="STREAM_TYPE"/>
    <xs:element ref="STREAM_TYPE_TO_PACKAGER"/>
    <xs:element ref="ORIGINATING_SYSTEM"/>
    <xs:element ref="DATE_TIME_CREATED"/>
    <xs:element ref="DATE_TIME_LAST_MODIFIED"/>
    <xs:element ref="FILE_ORGANIZATION"/>
    <xs:element ref="RECORD_FORMAT"/>
    <xs:element ref="RECORD_CONTROL"/>
    <xs:element ref="STREAM_SIZE_BYTES"/>
    <xs:element ref="MAXIMUM_RECORD_LENGTH_BYTES"/>
    <xs:element ref="FILE_NAME"/>
    <xs:element ref="CRC_TYPE"/>
    <xs:element ref="CRC"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="STREAM_TYPE_TO_PACKAGER" type="xs:NCName"/>
 <xs:element name="ORIGINATING_SYSTEM" type="xs:NCName"/>
 <xs:element name="DATE_TIME_CREATED" type="xs:NMTOKEN"/>
 <xs:element name="DATE_TIME_LAST_MODIFIED" type="xs:NMTOKEN"/>
 <xs:element name="FILE_ORGANIZATION" type="xs:NCName"/>
 <xs:element name="RECORD_FORMAT" type="xs:NCName"/>
 <xs:element name="RECORD_CONTROL" type="xs:NCName"/>
 <xs:element name="FILE_NAME" type="xs:NCName"/>
 <xs:element name="CANONICAL_STREAM_STRUCTURE">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="STREAM_TYPE"/>
    <xs:element ref="STREAM_RECORD_DELIMITER"/>
```

```
      <xs:element ref="STREAM_SIZE_BYTES"/>
      <xs:element ref="MAXIMUM_RECORD_LENGTH_BYTES"/>
      <xs:element ref="CRC_TYPE"/>
      <xs:element ref="CRC"/>
      <xs:element ref="RECOMMENDED_FILE_NAME"/>
      <xs:element ref="PROCESSING_REPORT"/>
      <xs:element ref="FORMAT_IDENTIFIER"/>
      <xs:element ref="ORDERED_APPLIED_ENCODINGS"/>
      <xs:element ref="ID_OF_ENCODED_FORMAT"/>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="STREAM_RECORD_DELIMITER" type="xs:NCName"/>
 <xs:element name="RECOMMENDED_FILE_NAME" type="xs:NCName"/>
 <xs:element name="PROCESSING_REPORT" type="xs:string"/>
 <xs:element name="FORMAT_IDENTIFIER" type="xs:NCName"/>
 <xs:element name="ORDERED_APPLIED_ENCODINGS" type="xs:NCName"/>
 <xs:element name="ID_OF_ENCODED_FORMAT" type="xs:NCName"/>
 <xs:element name="SUPPORTING_ATTRIBUTES">
  <xs:complexType/>
 </xs:element>
 <xs:element name="STREAM_TYPE" type="xs:string"/>
 <xs:element name="STREAM_SIZE_BYTES" type="xs:integer"/>
 <xs:element name="MAXIMUM_RECORD_LENGTH_BYTES" type="xs:integer"/>
 <xs:element name="CRC_TYPE" type="xs:NMTOKEN"/>
 <xs:element name="CRC" type="xs:NMTOKEN"/>
</xs:schema>
```

Annex B: XML Schema Best Practices Survey

Best Practices Item	***HP Best Practices***	ESA Best Practices	CNES Best Practices	DON Best Practices	EPA Best Practices	XFDU
1. When creating an XML schema, should you specify a targetNamespace?	Yes always.	If schema will be used in another schema, then try not to.	Yes	Yes	Yes, always	Yes
2. When creating an XML schema you need to specify a targetNamespace URI, what type of URI should you choose: a URN or a URL?	URI that resolves into HTTP URL			Must be URN	URN must be used	HTTP URL
3. Should I set the default namespace to the XML schema namespace or the targetNamespace?	The cleanest and simplest (but not the most compact) approach is to not use the default namespace and map both the XML schema namespace and the targetNamespace using prefixes	For known schemas, uses recommended prefixes (e.g. xsd). Set targetNamespace to be default if you have one			Declare namespace for XML schema, use xsd prefix. Don't use default namespace.	Uses prefixes, has targetNamespace, but not default.
4. When specifying an XML schema you often have the choice of placing a value in an attribute or an element, which should you use?	No consensus		No attributes.		Drifts more toward using elements but allows possibilities for attributes	Uses attributes for simple data, elements for complex data
5. Should local element names be qualified or unqualified?	**Element names should always be qualified.**	**Keep two s of schema: one with qualified another without; Use qualified for elements**		**Qualified**	**Element names must always be qualified.**	**No. elements are not qualified** **Would be fine to qualify elements**
6. Should all attributes in a document be namespace qualified?	No, attributeFormDefault should not be set; it should be left to default to "unqualified".	Use unqualified for attributes		Qualified	Use qualified for attributes	No. attributes are not qualified
7. When should I declare and use a global attribute?	• The attribute is used (or is being designed to be used) across several disjoint XML schemas and always has the same meaning. • You need to introduce a new attribute into another XML schema in order to extend that schema.	Minimize usage of global attributes	No attributes.		Prohibits usage of global attributes besides for metadata purposes	Uses global attributes for reuse in multiple types/elements

Best Practices Item	***HP Best Practices***	ESA Best Practices	CNES Best Practices	DON Best Practices	EPA Best Practices	XFDU
8. I have an attribute that is used several times in different elements in my XML schema. What should I do?	Create an attribute group to wrap the attribute in, and reference that attribute group from the complex types that contain the attribute.	Use attribute groups	No attributes.		Don't use attributes groups instead of elements, unless for metadata purposes.	Does just that
9. When should you define a global vs. a local element?	**Always define elements globally. Elements within model groups (choice, sequence) should always use the ref= form and never the type= form.**	**Depending on situation advocates usage of both; as well as mixed design.**	**Use types for structures that will be reused. Use local elements otherwise.**	**Mostly advocates usage of global elements**	**Advocates usage of global types and elements**	**Has several local elements. Most types are defined globally and attributes are then done as named elements of corresponding types. Using global elements resulted in bad code generation by JAXB** **Strong believe that type should be used.**
10. When should you define named complex types vs. inline anonymous complex types?	Complex types should always be named and never anonymous.	Use complex named types	Use types for structures that will be reused. Use local elements otherwise.	Advocates usage of global types	Advocates usage of global types	Uses globally defined types
11. When should you define named simple types vs. inline anonymous simple types?	**Elements and attributes should always use the type= form and never contain anonymous types.**				**Advocates usage of global simple types**	**Has some anonymous simple types** **If some local simple type is used for enumeration that is logically local, why make it a global type?**
12. Should I allow extension to my content model?	**Depends....**			**Discourages usage of substitutionGroups.**	**Allows for blockage of extension. Forbids usage of abstract datatypes and substitutionGroups**	**Uses substitutionGroups** **This technique appears to us as good semi-controlled technique to allow extension model.**
13. Should I allow arbitrary extension to my content model via <xs:any>?	Depends			Yes, but with namespace attribute		Not used
14. Should I use lax, strict, or skip for the processContent attribute?	Lax mostly					Now uses lax
15. Should I constrain my simple type to the exact data	Depends....		Not desirable			Not used

Best Practices Item	***HP Best Practices***	ESA Best Practices	CNES Best Practices	DON Best Practices	EPA Best Practices	XFDU
expected using facets, or should I leave room for extensibility?						
16. Should I set minOccurs and maxOccurs so that the data is as expected or should I allow a wider range of values to leave room for extensibility?	Generally, set the minOccurs and maxOccurs to what could reasonably make sense and document it.				Advocates usage of min/max occurs	Uses min/max occurs
17. If my data structure is a graph how should I represent that in my XML document and my XML schema?	**Use pointing mechanism via attributes or subelements, or key and keyref constructs.**	**Prefer key/keyref to ID/**		**Allows usage of ID/IDREF in certain cases**	**Advocates usage of key/keyref.**	**Uses ID,IDREF Theses constructs seem to be simple to manage and are supported by schema to object mapping toolkits.**
18. Should I use ID and IDREF?	**No, these DTD features have problems that prohibit general usage.**	**Prefer key/keyref to ID/IDREF**		**Allows usage of ID/IDREF in certain cases**	**Never**	**Uses ID,IDREF See #18**
19. Should I use unique, key, and keyref?	**Yes, these XML Schema constructs are excellent for describing relationships between values in your schema.**	**yes**		**Yes**	**Yes**	**No See #18**
20. How should you name elements and attributes?	Pick a descriptive name without it being excessively long. Use lower camel case	Avoid long names (more than 20 characters). Avoid making abbreviations removing vowels. Use upper camel case for elements, lower for attributes. Use noun to name elements	Avoid excessively long names which, while more meaningful, also increase the volume of data exchanged and do not necessarily assist legibility. If abbreviations used their use should be consistent. Only upper case used for elements and attribute values,. Use ALIAS attribute for	Lower camel case for attributes, upper camel case for elements. Avoid acronyms and abbreviations.	Use Upper Camel Case for elements and lower camel case for attributes. Avoid acronyms, capitalize them. Abbreviations shouldn't be used. Tag names should be concise.	Uses lower camel case mostly

Best Practices Item	*HP Best Practices*	ESA Best Practices	CNES Best Practices	DON Best Practices	EPA Best Practices	XFDU
			preserving the original name.			
21. How should you name simple and complex types?	The best practice is to append the word "Type" to all simple and complex type names		No obligation to append “Type”	Append word “Type”.	Append word “Type”	Type is mostly appended to type names
22. Should I use default values for attributes and elements?	No	No			No, but sometimes possible.	Used in one occasion.
23. Should I use fixed attributes?	No		No attributes.		No, but sometimes possible.	Not used
24. How should I version my XML schemas?	• **Major—completely different structure and semantics, most likely not backward compatible, new versions of applications are written to use new asset version** • **Minor—backward compatible changes which introduce new features without removing or changing the semantics of existing structures**	**Have version somewhere in schema; have version in instances; make older versions available; multiple approaches listed but no recommendation**		**Use version in namespace and use schema version attribute. Namespace must hold major/minor version. The attribute may also have subminor version (e.g. 1.0.3)**	**Must use XML schema version attribute to include major/minor version. Advocates having version in schema file name. Must have version attribute on the root of XML instances.**	**Has version attribute on top element. Has specificationVersion element. Currently discussed issue** **Topic of ongoing discussion**
25. How should I indicate the version in the namespace URI?	http://$domain/$groupSpecifier/$namespaceTitle/$year-$month			Use URN syntax with major and minor versions separated by a “:”	Use URN syntax with major and minor versions separated by a “:”	http://www.ccsds.org/xfdu/2004
26. Can I reuse a namespace for a new version of the schema?	For minor changes			For minor changes		Not used yet
27. Should I enable mixed content in my complexType Element?	No		No	Allows		Not used
28. How should I define a simple type which defines a possibly extensible set of enumeration values?	Using QNames					Uses string
29. Complex type code lists				Allows usage of code lists	Advocates usage of code lists.	Not used?
30. How should I specify	type substitution, <any>	Use wildcard.	Use sequence	Allows usage of	Sequence and choice;	Uses

Best Practices Item	***HP Best Practices***	ESA Best Practices	CNES Best Practices	DON Best Practices	EPA Best Practices	XFDU
an element which is a container of some set of elements?	wildcard or choice		and choice. No usage of substitutionGroups	choice	against usage of wildcard.	substitutionGroups
31. When should I use <all> model group?	Never		Never	Never	Never for data-centric schemas, sometimes for document-centric	Not used
32. Should I use complex type restriction in my XML schemas?	No.	Avoid	Avoid	Allows both restriction and extension. Instructs to use "final" on already restricted types to prevent further restriction.	Allows for both restriction and extension, as well as usage of "final" and "block" to stop extensibility of complex types.	Not used
33. Should I build up element content via multilevel subclassing or using composition?	Keep it simple, use composition (not more than 3,4 levels of subclassing)	Trees shouldn't be to long (and wide). Use composition (not more than 3 levels of subclassing)				Composition mostly used
34. Should I define a global attribute that will indicate to implementation the criticality of extension elements? (Must understand attribute)	**No**		**No attributes**			**Has one, but not currently used. Should discuss how to use it**
35. Should I make my elements nillable?	No			No		Not used
36. How should I define an element that is going to contain only simple content?	For maximum extensibility use the complexType with simpleContent form.			Use simpleContent in complexType		Not used
37. Schema header	**Yes**		**Yes**		**Must have.**	**Doesn't have one Should have one**
38. How should I handle a very large schema document?	The best practice is to break up your schema into logical sections and <include> these sections into the main XML schema file. Other namespaces should be <imported>.		By breaking and inclusion.	Strongly advocates usage of imports. Allows for usage of include in dev and runtime environments.	Advocates usage of include and import	Uses <import>
39. How can I indicate support for extension schemas?	You can either directly import the new extension schema into the latest version of your XML schema or create a new top level Schema					Not used yet

Best Practices Item	***HP Best Practices***	ESA Best Practices	CNES Best Practices	DON Best Practices	EPA Best Practices	XFDU
	document which includes your original XML schema and imports the extension XML schema[s].					
40. I can't express all the constraints that I need using XML Schema language, what should I do?	No agreement	Supplement with another schema language, software, etc.				Schematron is used via software.
41. Should I have some unique identifiers for schema components?		Have unique id for schema components (not attribute of ID type, just and attribute called "id")		Use unique identifiers. Use xsd:unique.	Use xsd:unique.	Not used
42. Should I use dangling type		**Yes**				**Not clear**
43. Types vs. elements		Use of types preferred	Advocates usage of type for reusability			Uses types mostly
44. Should schema location be specified when doing import?		**No**		**Yes**		**Imports Xlink with schema location specified.** **If it is not specified, then any xlink schema can be used, do we care?**
45. Binary attachments		For small sizes use base64 encoding; for large sizes use XOP				Allows base64 encoding but doesn't check the size. Intent to use XOP when implementation is available.
46. XML data compression		Use compression				Compression can be used outside of XFDU
47. Should Xlink/XPoiner be used?					No	Uses xlink
48. XML Encryption/Signing				Advocates where appropriate		Not used
49. Usage of ref			Ref must not be used			Only used in conjunction with substitutionGroups
50. Usage of recursive types			Forbidden			Used