XML Structure and Construction Rules
Draft Whitebook

05 May 2004

1 IIntroduction

This concept paper represents the beginning of a series of CCSDS Recommendations and Reports meant to augment the current CCSDS packaging recommendation (References [1], [2], [3], [4], [5], [6]),s to accommodate the current computing environment and meet evolving requirements.

1.1 PURPOSE AND SCOPE

The main purpose of this document is to define a staged set of CCSDS Recommendations for the packaging of data and metadata, including software, into a single package (e.g.,file or message) to facilitate information transfer and archiving. Another goal is to provide a detailed specification of core packaging structures and mechanisms that meets current CCSDS agency requrements and can be implemented to demonstrate practical, near-term results. This specification needs to be augmented with substantial proof-of-concept and performance prototyping of some of the basic packaging mechanisms in environments that include interactions with registries and repositories.

The scope of application of this document is the entire space informatics domain from operational messaging to interfacing with science archives. In recognition of this varied user community, this document proposes aggressive use of current and emerging W3C and Web Services standards to provide advanced data access techniques and adherence to the OAIS Reference Model (reference [7]) information model to provide improve support for long-term preservation of packaged information.

1.2 RATIONALE

The current CCSDS Standards for Data Packaging have not undergone a major revision in 15 years. The computing environment and the understanding of metadata have changed radically:

- Physical media →Electronic Transfer
  - The primary form of access to, and delivery of, both archived and recently produced data products has shifted from hard media to include substantia network delivery
- No standard language for metadata →XML
  - After 'bits' and 'ASCII', the language 'XML' can be viewed as the next universal data standard, as it has grown exponentially
- Homogeneous Remote Procedure Call→CORBA, SOAP
  - Communicating heterogeneous systems are increasingly using standard remote procedure calls or messaging protocols. The primary RPC and messaging protocol for the WWW is SOAP, an XML based protocol
- Little understanding of long-term preservation→OAIS RM
  - The The OAIS Reference Model has become a widely adopted starting point for standardization addressing the preservation of digital information. The OAIS defines and situates within functional and conceptual frameworks the concepts of Information Packages for archiving(Archival Information Packages, or AIPs), producer submission to an archive(Submission Information Packages, or SIPs), and archives dissemination to comsumers (Dissemination Information Packages, or DIPs).

- Record formats→Self describing data formats
    - Commenserate with XML, and rapidly growing computing power and storage capabililties, has been an increasing tendency to use data formats that are more self-describing.

Further, there are a number of new requirements that are needed in the Space domain to faciliate such functions as being able to describe multiple encodings of a data object, and to better describe the relationships among a set of data objects. Therefore it is necessary to define a new set of packaging standards while maintaining the existing functionality.

1.3 STRUCTURE OF THIS DOCUMENT

This document is divided into informative and normative chapters and annexes

Sections 1- 3 of this document are informative chapters that give a high level view of the rationale, the conceptual environment, some of the important design issues and an introduction to the terminology and concept

- Section 1 gives background to this effort, its purpose and scope, and a view of the overall document structure.
- Section 2 provides a high level view of the anticipated computing environment and the XFDU data model.
- Section 3 discusses some of the high level design decisions made in the specification of this Recommendation

Sections 4 –13 of this document are the normative portion of the specification

- Section 4, entitled "Packaging Techniques" is transition from informative to normative sections. It provides a description and an XML schema diagram of the first level elements of the XFDU packaging specification material that is proposed to be the basis for the White Book. It also discusses the “utility” types that are reused many times within the XFDU schema.

Sections 5-12 presents a detailed breakdown of the important entities represented in the schema. Each section is organized in the following manner:

- N.1 –overview
- N.2 – XML schema and XML SPY diagram
- N.3 – XML Examples
- Semantics and Issues

Section 13 is the full XML Schema for the Version 1 XFDU. In the case of differences between the full Schema in Section and the narratives and partial schemas in prior chapters the full XML Schema is the ruling specification.

Annexes 1-6

- Annex 1 provides a Unified Modelling Language (UML) view of the overall XFDU.
- Annex 2 provides the full examples that are are the source of the examples in Sections 5-12
- Annex 3 provides a summary of relevant external standards.
- Annex 4 identifies some use cases.

- Annex 5 dentifies requirements that have been derived from use cases and actual experience with the current CCSDS packaging standards.

1.4 TERMS –NEEDS TO BE ENHANCED

The terms used in this document and their meanings are listed below.

Component
: Refers to a file that can be grouped together to be part of a Collection, or Package.

Collection
: Refers to Components that are gathered together along with a Manifest. This is analogous to files on a file system.

Association
: Refers to a relationship between Components in a Collection, or other metadata related to a Component or the Collection as a whole.

Manifest
: A document containing metadata about Components, and the Associations between them. This information is stored as a Component, using an XML language designed for just this purpose.

Package
: Refers to a Collection that is bundled together, or packaged, into one file using a defined packaging scheme. All Packages are Collections, but not all Collections have been packaged, so they are not all Packages.

1.5 REFERENCES

[1]*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. (Also as ISO 12175)

[2] *ASCII Encoded English (CCSD0002)*. Recommendation for Space Data System Standards, CCSDS 643.0-B-1. Blue Book. Issue 1. Washington D.C.: CCSDS, November 1992. (Also ISO 14962)

[3] *Standard Formatted Data Units — Control Authority Procedures*. Recommendataion for Space Data Systems Standards, CCSDS 630.0-B-1. Blue Book. Issue 1. Washington, D.C., CCSDS, June 1993. (Also 13764)

[4] *Standard Formatted Data Units — Control Authority Data Structures*. Recommendation for Space Data System Standards, CCSDS 632.0-B-1. Blue Book. Issue 1. Washington D.C.: CCSDS, November 1994. (Also ISO 15395)

[5] *Standard Formatted Data Units-Referencing Environment*. Recommendation for Space Data System Standards, CCSDS 622.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 1997. (Also ISO 15888)

[6] *Parameter Value Language Specification (CCSD0006 and CCSD0008).* Recommendation for Space Data System Standards, CCSDS 641.0-B-2. Blue Book. Issue 2. Washington D.C.: CCSDS, June 2000. (Also ISO 14961)

[7] *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. (Also ISO 14721)

[8] Metadata Encoding and Transmission Standard (METS)

[9] XPACK

[10] Soap with Attachments

[11] Direct Internet Message Encapsulation (DIME)

2 Overview of Proposed Packaging Structure

The section provides an overview of some of the key concepts that are incorporated in the design of the XFDU packaging recommendation.

2.1 ENVIRONMENT

Figure 1 is illustrates an abstract package in a generic computing environment to provide a basis for discussion of concepts relevant to this document. The focus of this diagram is a collection of physical file that have been bundled together due to some interrelationship. In this case this collection of files have been bundled together into a single container called a Package Interchange File and an XML document called a manifest has been included to document the relations among and index the locations of the various files containing data and metadata. The figure also shows external resources including other packages, file systems and repositories. In an environment with sufficient connectivity, reliability, and bandwidth the package could include pointers to resources outside of the package. The resolution of these pointers would be a software service provided by the data producer or a software toolkit at the data consumer site.

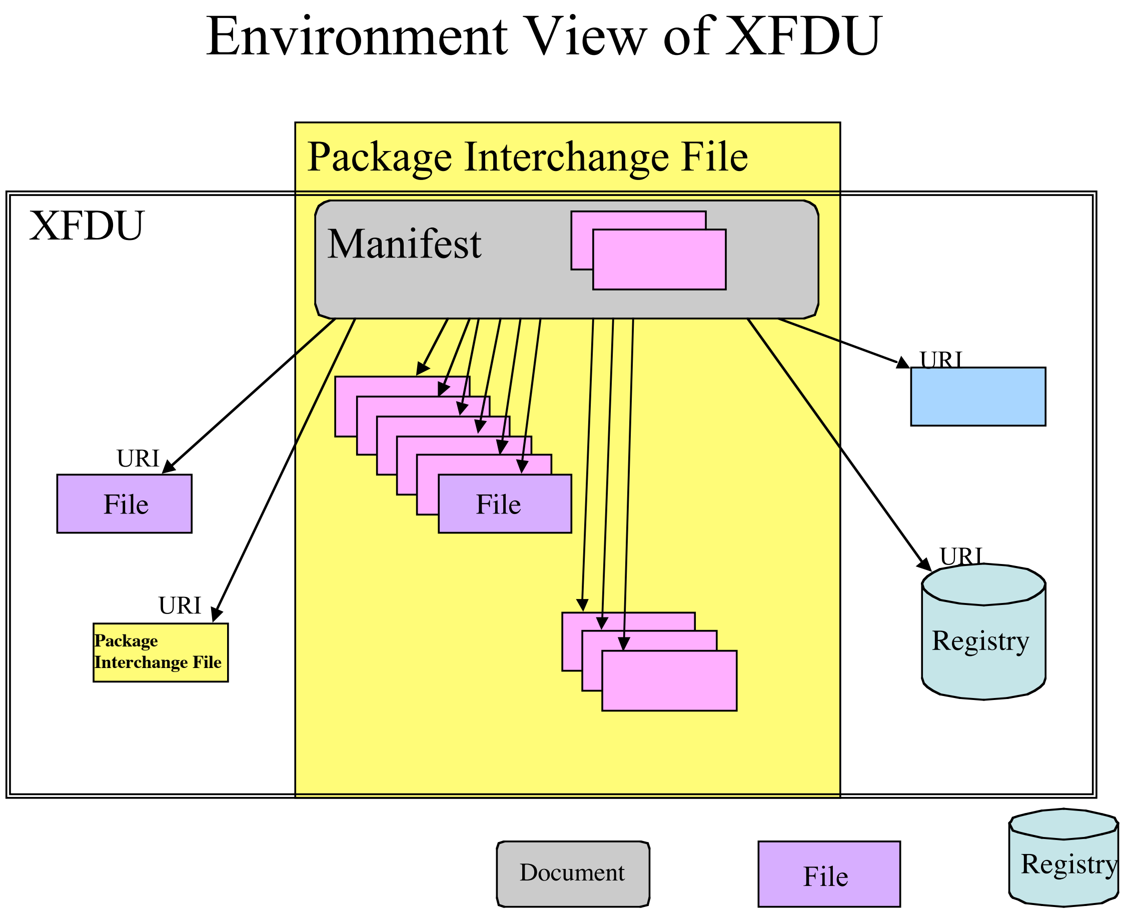
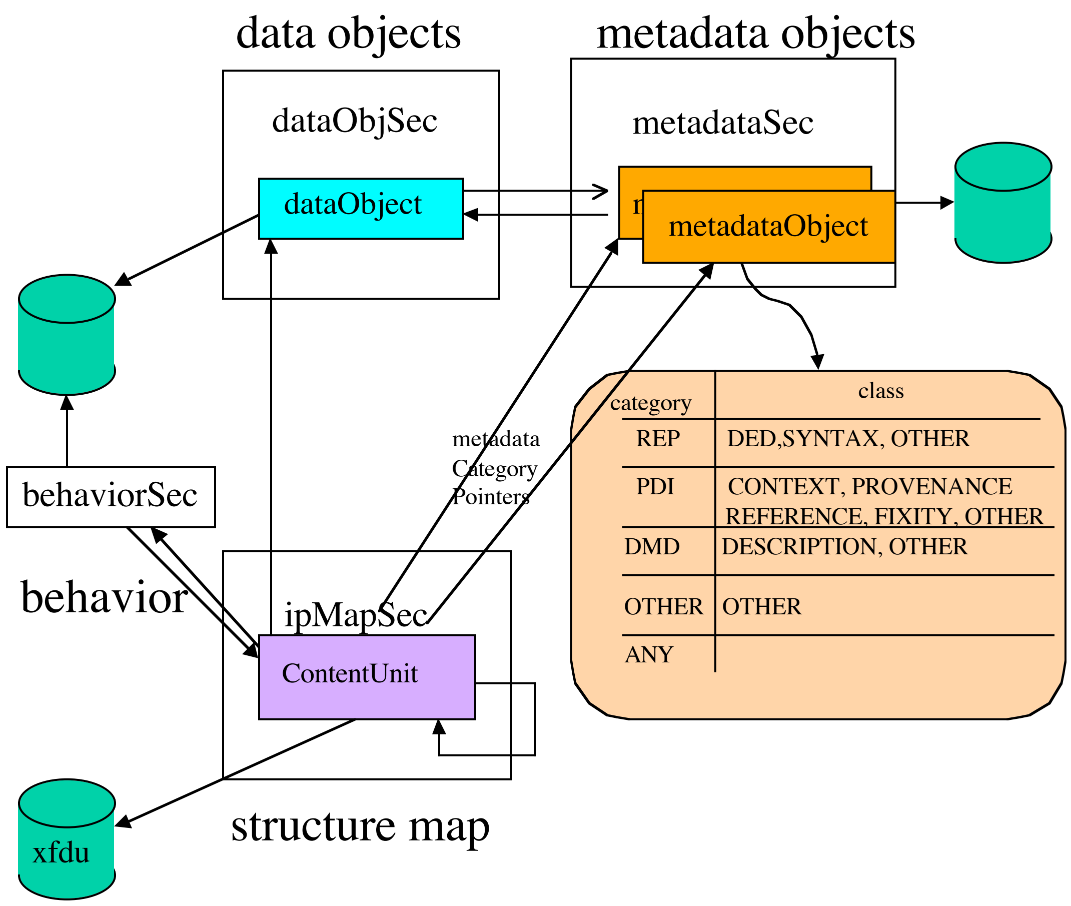


Figure :Environment/Conceptual View of XML Packaging

2.2 LOGICAL STRUCTURE

Figure 2 provides an expanded view of the XML manifest file showing the key entities and the possible references among them. The arrowheads show the direction of the references (e.g., the contentUnit entity references the dataObject entity). The contentUnit provides the primary view into the package as it refers to each of the data objects and it associates appropriate metadata with each data object. The reference to the metadata is via one or more metadata Category pointers. For each such pointer, there are a set of metadata classes that may be chosen to further classify the metadata object. The actual metadataObject may be included in the manifest file or referenced by URI They may also contain other contentUnits in the form of an XFDU package. The figure also introduces the names and XML Labels for some of the XML entities that are discussed in the next section



category	class
REP	DED,SYNTAX, OTHER
PDI	CONTEXT, PROVENANCE REFERENCE, FIXITY, OTHER
DMD	DESCRIPTION, OTHER
OTHER	OTHER
ANY	

Figure 1

3 Phased Release Design Decisions

The section provides an overview of some of the key concepts that are incorporated in the design of the XFDU packaging recommendation. The major issues addressed include the extent of reuse of the concepts from METS[REF1] and XPACK[REF2], the underlying packaging mechanisms, the use of the OAIS information model and the functionality desired in staged releases. A summary of the decision making process and more detail on the Related efforts can be found in Annex 3 and 4

Support of Physical Packaging

XFDU version 1 must support the ZIP and single document forms. The SOAP/MIME/DIME forms should be prototyped but the underlying protocols may not be stable in the version 1 timeframe.

Mandatory Manifest/Table of Contents

Version 1 of the XFDU is to have a mandatory XML encoded map of the information package and metadata to enable a common schema for all packaging forms.

Flexible linkage of Manifest to Data and Metadata

The XFDU Version 1 Manifest should:

- Data Objects that are contained in the manifest are to be encoded in base64 or XML
- Data Objects that are included by reference from the manifest are to exist as files in the XFDU package or as files with known URIs either in a repository or in a location accessable via URL
- Metadata objects that are contained in the manifest are to be encoded in base64 or XML
- Metadata objects that are included by reference from the manifest are to exist as files in the XFDU package or as files with known URIs either in a repository or in a location accessable via URL
- Information Objects can reference applicable Metadata objects by ID where the name of the referencing attribute is used to catagorize the Metadata
- Metadata objects may be treated as data objects. This enables direct mapping to the OAIS representation net where each metadata object is an information object containing both data object and representation information

.

3.1.1 RELATIONSHIP DESCRIPTION

A major component of information packaging is the description of the relationships among the various metadata and data objects in both human readable and machine interpretable forms. The “Semantic Web” efforts have led to several XML notations for expressing relationships and rules among resources. The two main efforts reviewed in this area were Resource Description Format (RDF) and Web Ontology Language (OWL). RDF has been a World Wide Web Consortium (W3C) Recommendation for several years but software support and automated applications have lagged the recommendation considerably. OWL has recently been promoted to a W3C Candidate Recommendation. Based on experience with other complex W3C Recommendations, this means that a final recommendation is expected in 1-2 years. However, OWL is derived from several existing ontology languages, so there is significant legacy software that can quickly be modified to the OWL syntax.

Due to the lack of mature XML software products in this area, the initial version of this recommendation will not use a formal XML grammer to express the range of potential relationships. Instead, the XFDU schema uses the OAIS information model to classify information objects and content unit types to define the overall structure and semantics of contained objects. Alsothe complete Xlink schema is included in the XFDU referencing mechanisms to allow the use of complex Xlink attribues to model resource relationships.

It is anticipated that Future versions of this recommendation will use an XML based language such as OWL to improve description and enable software services for generic relationships.

3.1.2 EXTENSIBILITY MECHANISMS

In order to allow an orderly evolution of the XFDU schema and to allow the development of specialized versions of complex elements to enable additional functionality or alternative implementations during prototyping
There are a number of XML Schema mechanisms that can be used to achieve the goal:

1. Use of an abstract element and element substitution using substitutionGroup
2. Use of abstract type and type substitution using xsi:type
3. Use of a <choice> element
4. Use of a dangling type

The use of dangling types is theoretically the most flexible solution, but is not supported by most XML parsers. The use of choice is the least attractive solution because it does not support the use of complex elements and creates rigid schemas.

Type substitution is slightly superior to element substitution from a schema design viewpoint; however, there are two factors that inform the decision to use element substitution for this version. The first is the visibility of xsi:type in every XML instance. This has proven unpopular with users of other standard schemas. The second factor is that some early experimentation with inputting XML Schema to JAVA tools revealed that element substitution was partially supported while type substitution was not supported.

Annex D contains a more thorough discussion of this issue and a reference to a complete analysis.

3.2 STAGED RELEASE FUNCTIONALITY

3.2.1 VERSION 1 SHOULD INCLUDE:

- The capabilities of current SFDU packaging and CCSDS Control Authority Concepts
- Support for data descriptions, MIME types, self describing formats, and detached data descriptions
- Support for the OAIS RM Information Model Concepts and Types
- Flexible linkage to Metadata
- The ability to encapsulate related files/resources into a single file/container
- The ability to reference both content and metadata resources contained in the same container or at a known URL
- The ability to allow/reverse multiple transformations on files
- Behaviors
    - Web Service Interfaces
    - Portable Code

3.2.2 VERSION 2 FUNCTIONALITY ENABLED

- Behavior
    - Automatic execution
    - Scripting (Output of one behavior as input to another behavior)
- Software Updates
- Process Definition
- Relationship Definition

4 XFDU Complex Type

4.1 OVERVIEW

There are seven sections that may appear in an XFDU document (i.e. Manifest). A high level XML Spy[REF3] diagram of the XFDU is shown as Figure 3:

1. **Package Header** (packHeader): Administrative metadata for the whole XFDU, such as version, operating system, hardware, author, etc, and metadata about transformations and behaviours that must be understood

2. **Metadata Section (**MetadataSec): This section records all of the metadata for all items in the XFDU package. Multiple metadata objects are allowed so that the metadata can be recorded for each separate item within the XFDU object. The metadata schema allows the package designer to define any metadata model by providing attributes for both metadata categories and a classification scheme for finer defintion within categories. The model provides predefined metadata categories and classes via enumerate attributes that follow the OAIS information model as follows:

   - Descriptive information is intended for the use of Finding Aids such as Catalogs or Search Engines.
   - The Representation Section and its subsections, syntax information (syntaxMd), static semantics (dedMd), and unclassified metadata (otherMd)
   - The classification of the PDI Section - reference, context, provenance, and fixity
   -

3. **Information Package Map Section (ipMapSec)** outlines a hierarchical structure for the original object being encoded, by a series of nested **contentUnit** elements. Content units contain pointers to the data objects and to the metadata associated with those objects .

4. **Data Object Section (dataObjectSec)** contains a number of dataObjEntry elements. A Data Object Entry contains some file content and any data required to allow the information consumer to reverse any transformations that have been performed on the object and restore it to the byte stream intended for the original designated community and describedby the Representation metadata in the Content Unit

5. **Behavior Section (behaviorSec)** can be used to associate executable behaviors with content in the XFDU object. A behavior section contains an interface definition element that represents an abstract definition of the set of behaviors represented by a particular behavior section and a mechanism section also has a behavior mechanism that is a module of executable code that implements and runs the behaviors defined abstractly by the interface definition.

4.2 XML SCHEMA

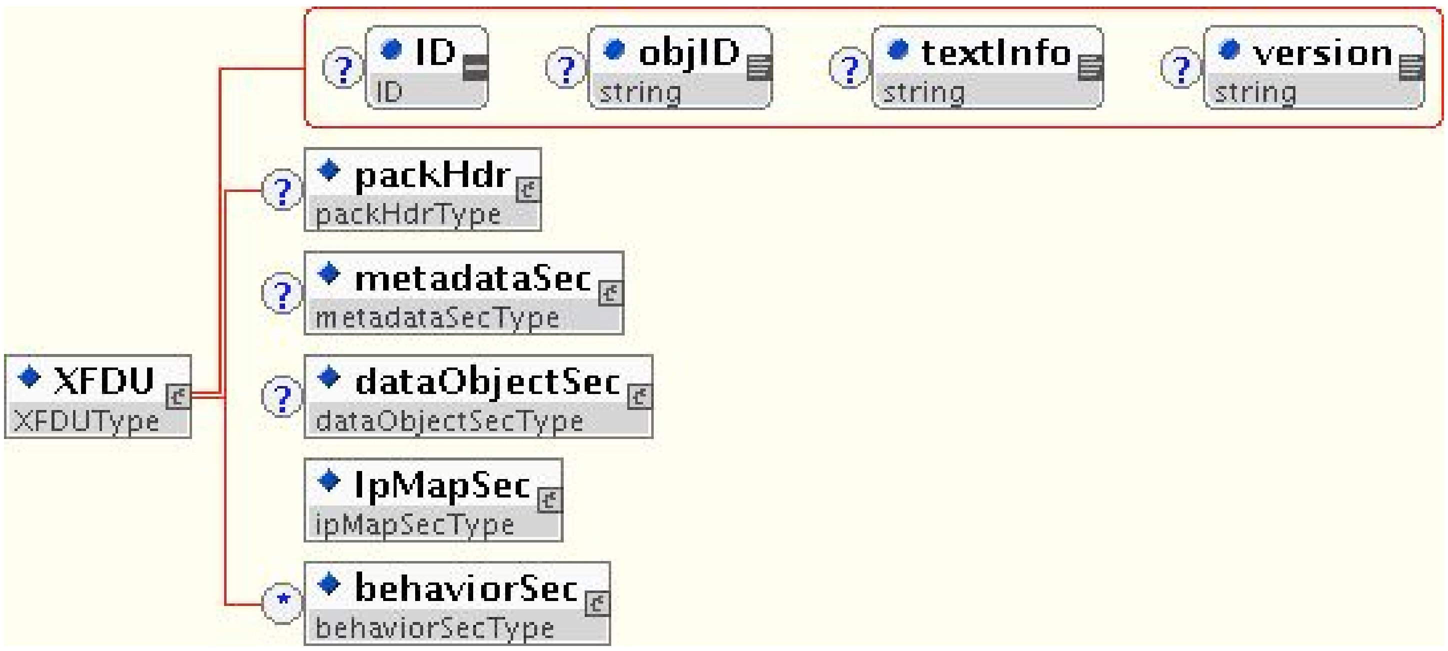


Figure 2

4.3 UTILITY TYPES

These classes are reused throughout the XFDU schema to represent some recurring structures:

- ReferenceType - Entity that can reference (have link) to other resource.
- The infoObjEntryPtr is an empty element that refernces dataObjects that represent the containing Information Object
- FcontentType - Entity that encapsulates either binary or XML arbitrary content.
  - BinData - Entity that encapsulates base64 encoded data(e.g. binary data)
  - XmlData - Entity that encapsulates 1 to many pieces of arbitrary XML data

4.3.1 XML SCHEMAS

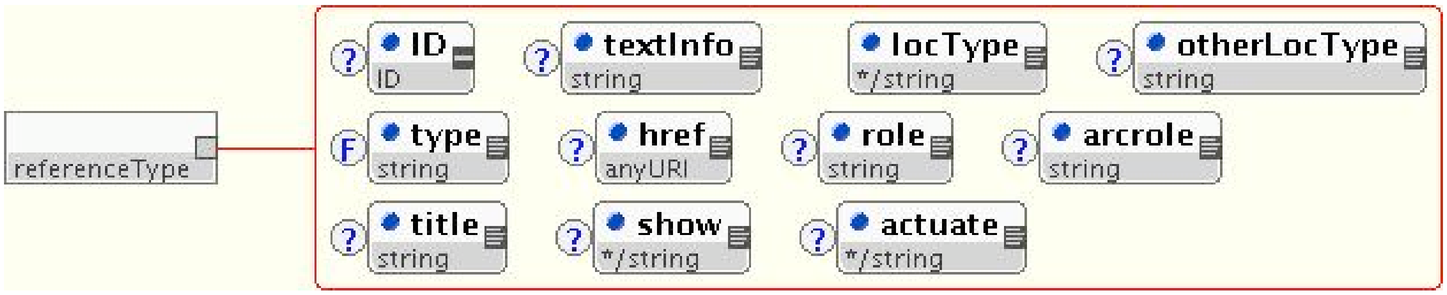


Figure 4

```
<xsd:complexType name = "referenceType">
  <xsd:attribute name = "ID" type = "xsd:ID"/>
  <xsd:attributeGroup ref = "LOCATION"/>
  <xsd:attributeGroup ref = "xlink:simpleLink"/>
</xsd:complexType>
```

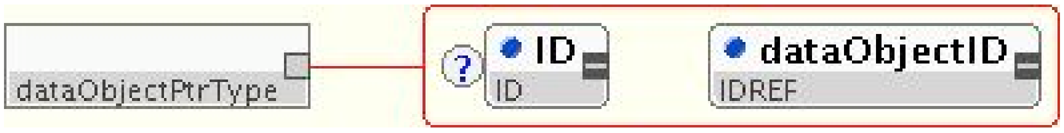


Figure 5

```
<xsd:element name = "infoObjEntryPtr">
  <xsd:annotation>
<xsd:documentation>The infoObjEntryPtr is an empty element that refernces dataObjects that represent the containing
the  dataObjectID: The infoObjEntryPtr has two attributes: 1. ID: an XML ID for this element; and 2.
 dataObjectID: an IDREF to a dataObject element reference contentUnit containing this datatpr.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:attribute name = "ID" type = "xsd:ID"/>
    <xsd:attribute name = "infoObjEntryID" type = "xsd:IDREF"/>
  </xsd:complexType>
</xsd:element>
```

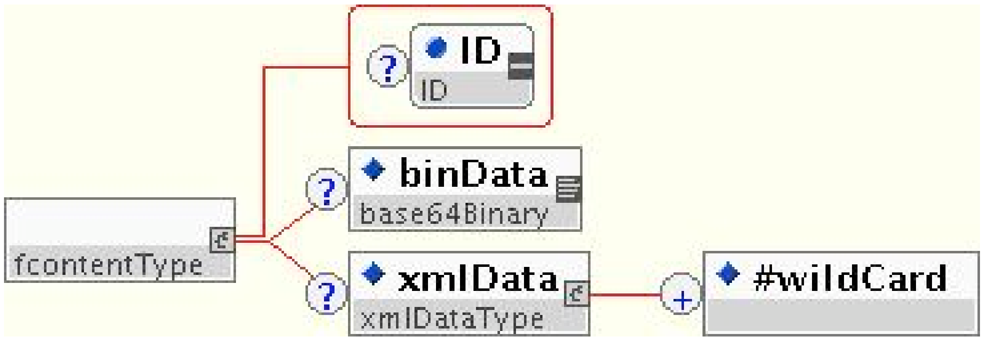


Figure 6 fcontentType/binData/xmlData

```
<xsd:complexType name = "fcontentType">
  <xsd:annotation>
    <xsd:documentation>
      fContentType encapsulates and agregates a type that can have a choice of either
      binary or xml data
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice>
    <xsd:element ref = "binData" minOccurs = "0"/>
    <xsd:element ref = "xmlData" minOccurs = "0"/>
  </xsd:choice>
  <xsd:attribute name = "ID" type = "xsd:ID"/>
</xsd:complexType>

          <xsd:element name = "binData" type = "xsd:base64Binary">
            <xsd:annotation>
              <xsd:documentation>A wrapper to contain Base64 encoded metadata.</xsd:documentation>
            </xsd:annotation>
          </xsd:element>


          <xsd:element name = "xmlData">
            <xsd:annotation>
              <xsd:documentation>A wrapper to contain XML-based metadata.</xsd:documentation>
            </xsd:annotation>
            <xsd:complexType>
              <xsd:sequence>
                <xsd:any namespace = "##any" processContents = "strict" maxOccurs = "unbounded"/>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
```

5 Packaging Header Type

5.1 OVERVIEW

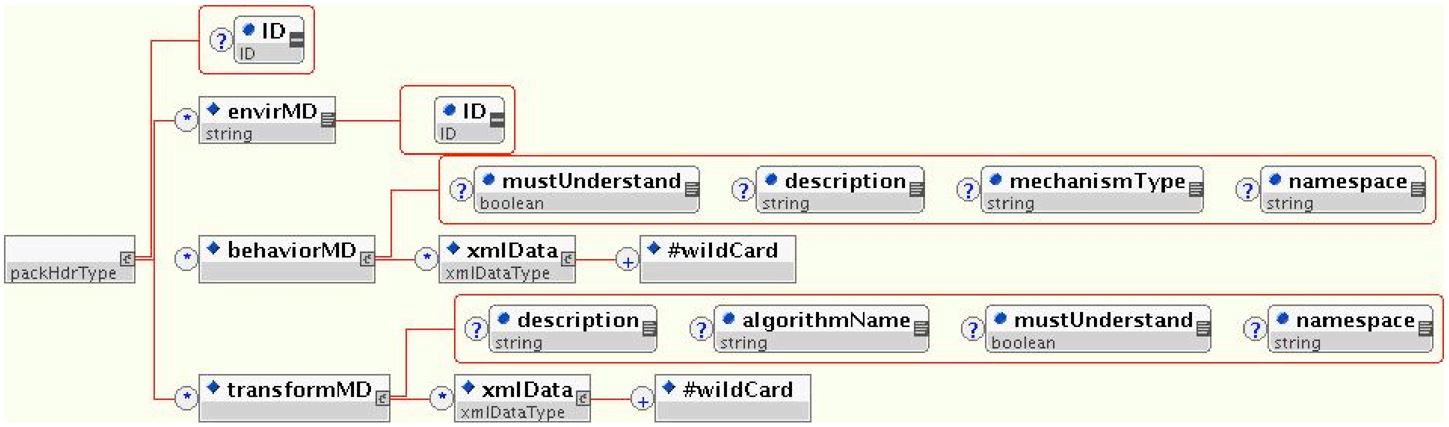
The Package Header is an XML Complex Type for administrative and technical requirements metadata that apply to the containing XFDU package. The package header section consists of three optional sections:

- EnvirMD: specification of the hardware and software platform which created this package and other administrative data such as creator and modification authority which applies to the whole instance of the XFDU
- behaviorMD: mustunderMD is based on the SOAP mustunderMD element and is a list of namespace elements that are required to access the Content of this XFDU instance. Used for versioning issues
- transformMD the names, classifications, parameter names/types and any other information needed to reverse transformations used in the XFDU).

PackHdrType has a single attribute, ID (XML ID).

The Packaging Header should be a self contained entity (containing both its schema and data) so it may be process independently from the main body of the XFDU.

5.2 XML SCHEMA SYNTAX



```
<xsd:complexType name = "packHdrType">
<xsd:annotation>
  <xsd:documentation>packHdrType: Complex Type for metadata about the
  mapping of the logical packages to the physical structures. The
  package header section consists of three possible subsidiary
  sections: envirMD (specification of the hardware and software
  platform which created this package), behaviorMD (behavior mechanism
  related metadata), and transformMD (the names, classifications, parameter
  names/types and any other information needed to reverse
  transformations used in the XFDU). Both transformMD and behaviorMD have an optional
  mustUnderstand attribute that declares if the reader of this package must
  understand described transformation, behavior mechanisms in order to
  process content of the package. packHdrType has a single
  attribute, ID: an XML ID.
```

```
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name = "envirMD" minOccurs = "0" maxOccurs = "unbounded">
        <xsd:annotation>
          <xsd:documentation>envirMD: technical metadata. The envirMD element
          provides a wrapper around a generic metadata section that
          should contain technical metadata regarding a dataObject or dataObjects. It
          has a single attribute, ID, which dataObject/dataObjectGrp elements can use
          to reference the technical metadata that applies to them.
          </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base = "xsd:string">
              <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name = "behaviorMD" minOccurs = "0" maxOccurs = "unbounded">
        <xsd:annotation>
          <xsd:documentation>
            behaviorMD contains:
            mustUnderstand - indicates if this mechanism must be understood by processor
            description - general description
            mechanismType - type of behavior mechanism (e.g. WS,ANT,JAVA) (should be made into enumeration most
likely)
            namespace - namespace of the specified technology if any
            behaviorMD has an optional xmlData element to include any additional metadata if needed
          </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name = "xmlData" type = "xmlDataType" minOccurs = "0" maxOccurs = "unbounded"/>
          </xsd:sequence>
          <xsd:attribute ref = "mustUnderstand"/>
          <xsd:attribute name = "description" type = "xsd:string"/>
          <xsd:attribute name = "mechanismType" type = "xsd:string"/>
          <xsd:attribute ref = "namespace"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name = "transformMD" minOccurs = "0" maxOccurs = "unbounded">
        <xsd:annotation>
          <xsd:documentation>transformMD (the names, classifications, parameter
          names/types and any other information needed to reverse
          transformations used in the XFDU)
          transformMD contains:
          mustUnderstand - indicates if this transformation technology must be understood by processor
          description - gneral description
          algorithmName -name of transformation algorithm
          namespace - namespace of the specified technology if any
          transformMD has optional xmlData element to include any additional metadata if needed
          </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name = "xmlData" type = "xmlDataType" minOccurs = "0" maxOccurs = "unbounded"/>
          </xsd:sequence>
          <xsd:attribute name = "description" type = "xsd:string"/>
          <xsd:attribute name = "algorithmName" type = "xsd:string"/>
          <xsd:attribute ref = "mustUnderstand"/>
          <xsd:attribute ref = "namespace"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name = "ID" type = "xsd:ID"/>
  </xsd:complexType>
```

5.3 EXAMPLES

5.3.1 HEADER USING MUSTUNDERSTAND

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XFDU>
   <packHdr>
      <envirMD ID="envirMD1">All platforms</envirMD>
      <transformMD algorithmName="blowfish" mustUnderstand="true"
        description="encryption"/>
   </packHdr>
```

5.4 SEMANTICS AND ISSUES

1. Do we want to define vocabularies for use in the environment section

6 Information Package Map

6.1 OVERVIEW

The Information Package Map Section (ipMapSec) outlines a hierarchical structure for the original object being encoded, bya series of nested contentUnit elements. The ipMapSec element has the following attributes: ID: an XML ID for the element; TYPE: the type of Information Product provided. Typical values will be"AIP" for a map which describes a complete AIP obeying all constrainsts and cardinalitiies in the OAIS reference model "SIP" for a map which describes a Submission Information Package TEXTINFO: a string to describe the ipMapSec to users

6.2 XML SCHEMA

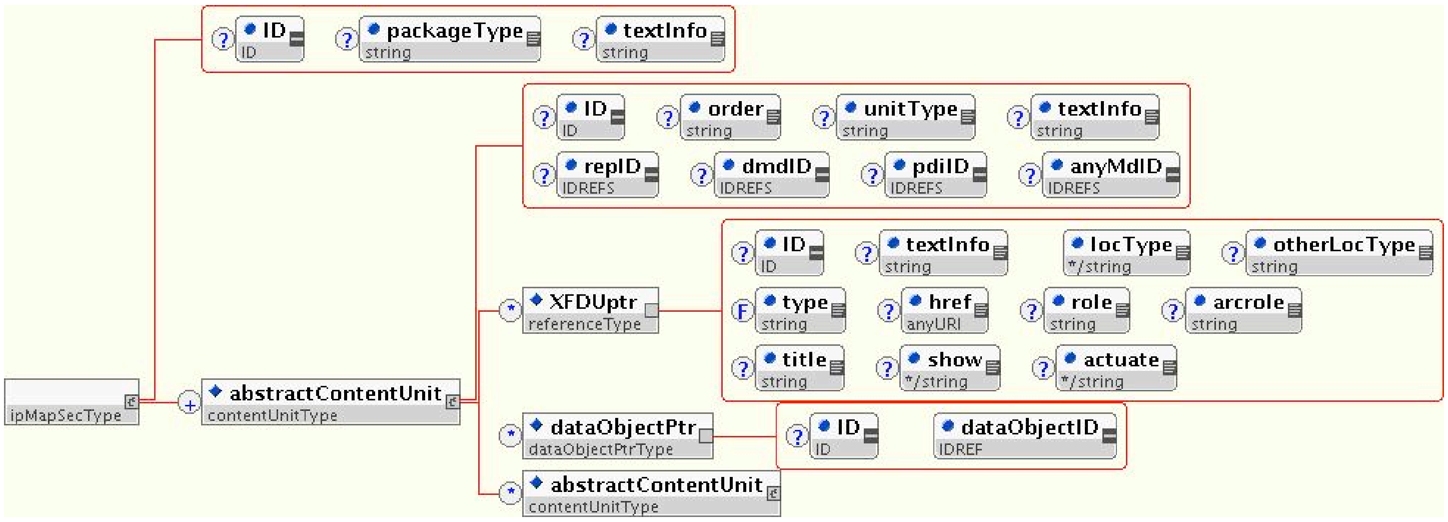


Figure 8

```
<xsd:complexType name = "ipMapSecType">
  <xsd:annotation>
    <xsd:documentation>ipMapSecType Complex Type The Information Package Map
      Section (ipMapSec) outlines a hierarchical structure for the
      original object being encoded, using a series of nested
      contentUnit elements. An element of ipMapSecType has the following
      attributes: ID: an XML ID for the element; TYPE: the type of
      Information Product provided. Typical values will be"AIP" for a
      map which describes a complete AIP obeying all constrainsts and
      cardinalitiies in the OAIS reference model "SIP" for a map which
      describes a Submission Information Package textInfo: a string to
      describe the ipMapSec to users. packageType: a type for the object, e.g., book, journal, stereograph, etc.;
      Concrete implementation of contentUnit (defaultContentUnit, behavioralContentUnit, etc)
```

```
        have to be used in the instance document.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref = "abstractContentUnit" maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" type = "xsd:ID"/>
    <xsd:attribute name = "packageType" type = "xsd:string"/>
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
  </xsd:complexType>
```

6.3 EXAMPLES

6.3.1 MULTIPLE CONTENT SECTIONS

```
  <IpMapSec>
    <contentUnit repID = "atdMD" pdiID = "provenance" dmdID = "ECSDMD">
      <dataObjectPtr dataObjectID = "mpeg21"/>
      <contentUnit order = "1" textInfo = "Root content unit for HDF data">
        <contentUnit order = "1.1" pdiID = "provenance" textInfo = "content unit for hdfFile0" dmdID = "ECSDMD">
          <dataObjectPtr dataObjectID = "hdfFile0"/>
        </contentUnit>
        <contentUnit order = "1.2" pdiID = "provenance" textInfo = "content unit for hdfFile1" dmdID = "ECSDMD">
          <dataObjectPtr dataObjectID = "hdfFile1"/>
        </contentUnit>
        <contentUnit order = "1.3" pdiID = "provenance" textInfo = "content unit for hdfFile2" dmdID = "ECSDMD">
          <dataObjectPtr dataObjectID = "hdfFile2"/>
        </contentUnit>
      </contentUnit>
      <contentUnit textInfo = "content unit for orbit data">
        <dataObjectPtr dataObjectID = "orbitalData"/>
      </contentUnit>
    </contentUnit>
    <contentUnit textInfo = "content unit ATD metadata">
      <dataObjectPtr dataObjectID = "ATDMD"/>
    </contentUnit>
  </IpMapSec>
```

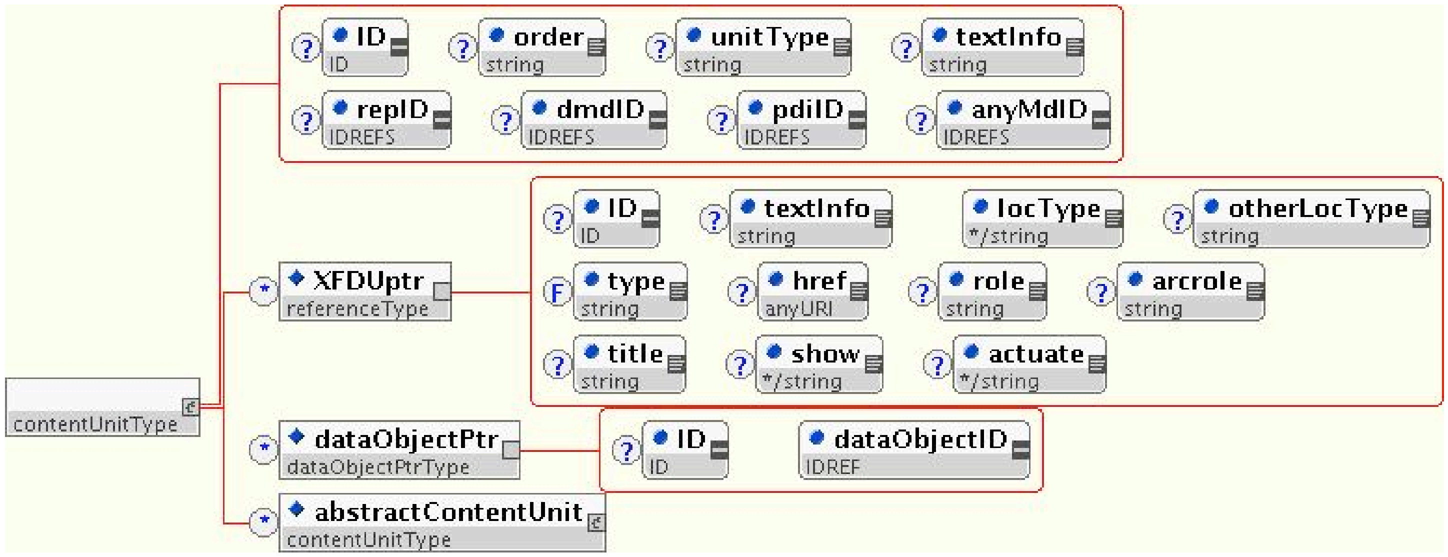
7 Content Units

7.1 OVERVIEW

AbstractContentUnit Complex Type The XFDU standard represents a data package structurally as a series of nested content units, that is, as a hierarchy (e.g., a data product, which is composed of datasets, which are composed of time series, which are composed of records). Every content node in the structural map hierarchy may be connected (via subsidiary xfduptr or infoptr elements) to information objects which represent that unit, A portion of the whole package. The content units element has the following attributes:

- ID (an XML ID);
- ORDER: an numeric string (e.g., 1.1, 1.2.1, 3,) representation of this Units order among its siblings (e.g., its sequence);
- TEXTINFO: a string label to describe this contentUnit to an end user viewing the document, as per a table of contents entry
- REPID: a set of IDREFs to representation information sections within this XFDU document applicable to this contentUnit
- DMDID: a set of IDREFS to descriptive information sections within this XFDU document applicable to this contentUnit.
- PDIID: a set of IDREFS to preservation description information sections within this XFDU document applicable to this contentUnit
- TYPE: a type of content unit (e.g., Application Data Unit, Data Description Unit, Software Installation Unit,etc.).. The Type attribute indicates the semantics of the Content and implies many of the relationships among the contained Content Objects

7.2 XML SCHEMA



```
<xsd:complexType name = "contentUnitType">
  <xsd:annotation>
    <xsd:documentation>ContentUnit Complex Type The XFDU standard
    represents a data package structurally as a series of nested
    content units, that is, as a hierarchy (e.g., a data product,
    which is composed of datasets, which are composed of time
    series, which are composed of records). Every content node in
    the structural map hierarchy may be connected (via subsidiary
    XFDUptr or dataObjectPtr elements) to information objects which
    represent that unit as a portion of the whole package. The content
    units element has the following attributes:
    1.ID (an XML ID);
    2.order: an numeric string (e.g., 1.1, 1.2.1, 3,) representation
    of this unit's order among its siblings (e.g., its sequence);
    3.textInfo: a string label to describe this contentUnit to an end
    user viewing the document, as per a table of contents entry
    4.repID: a set of IDREFs to representation information sections
    within this XFDU document applicable to this contentUnit.
    5.dmdID: a set of IDREFS to descriptive information sections
    within this XFDU document applicable to this contentUnit.
    6.pdiID: a set of IDREFS to preservation description information
    sections within this XFDU document applicable to this
    contentUnit
    7.anyMdID: a set of IDREFS to any other metadata sections that do not fit
    rep,dmd or pdi metdata related to this  contentUnit
    8.unitType: a type of content unit (e.g., Application
    Data Unit, Data Description Unit, Software Installation Unit, etc.).
    contentUnitType is declared as a base type for concrete implementations of contentUnit.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name = "XFDUptr" type = "referenceType" minOccurs = "0" maxOccurs = "unbounded">
      <xsd:annotation>
        <xsd:documentation>XFDUptr:XFDU Pointer. The XFDUptr element allows a
        content unit to be associated with a separate XFDU containing
        the content corresponding with that contentUnit, rather than
        pointing to one or more internal dataObjects. A typical instance of
        this would be the case of a thematic data product that collects
        data products from several instruments observe an event of
        interest. The content units for each instrument datasets might
        point to separate XFDUs, rather than having dataObjects and dataObject
        groups for every dataset encoded in one package. The XFDUptr
        element may have the following attributes: ID: an XML ID for
        this element locType: the type of locator contained in the
        xlink:href attribute; otherLocType: a string to indicate an
        alternative locType if the locType attribute itself has a value
        of "OTHER." xlink:href: see XLink standard
        (http://www.w3.org/TR/xlink) xlink:role: "" xlink:arcrole: ""
        xlink:title: "" xlink:show: "" xlink:actuate: "" NOTE: XFDUptr
        is an empty element. The location of the resource pointed to
        MUST be stored in the xlink:href element.

        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name = "dataObjectPtr" type = "dataObjectPtrType" minOccurs = "0" maxOccurs = "unbounded"/>
    <xsd:element ref = "abstractContentUnit" minOccurs = "0" maxOccurs = "unbounded"/>
  </xsd:sequence>
  <xsd:attribute name = "ID" type = "xsd:ID"/>
  <xsd:attribute name = "order" type = "xsd:string"/>
```

```
    <xsd:attribute name = "unitType" type = "xsd:string"/>
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
    <xsd:attribute name = "repID" type = "xsd:IDREFS"/>
    <xsd:attribute name = "dmdID" type = "xsd:IDREFS"/>
    <xsd:attribute name = "pdiID" type = "xsd:IDREFS"/>
    <xsd:attribute name = "anyMdID" type = "xsd:IDREFS"/>
  </xsd:complexType>
<xsd:element name = "abstractContentUnit" type = "contentUnitType" abstract = "true">
    <xsd:annotation>
      <xsd:documentation>abstractContentUnit is abstract implementation of
      contentUnitType. It cannot be instantiated in the instance
      document. Instead, concrete implementations would have to be
      used which are declared part ofcontentUnit substitutionGroup
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref = "abstractContentUnit" maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" type = "xsd:ID"/>
    <xsd:attribute name = "packageType" type = "xsd:string"/>
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
  </xsd:complexType>
```

Figure 9

7.3 EXAMPLES

7.3.1 SPECIALIZATION OF CONTENT UNIT USING SUBSTITUTION GROUPS

7.3.1.1 Example of schema extension via substitution group

```
  <xsd:element name = "contentUnit" type = "contentUnitType" substitutionGroup = "abstractContentUnit">
    <xsd:annotation>
      <xsd:documentation>contentUnit is basic concrete
        implementation of abstract conentUnit. Its instace can be used
        in the instance document in the place where contentUnit declared
        to be present.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:element name = "behavioralContentUnit" substitutionGroup = "contentUnit">
    <xsd:annotation>
      <xsd:documentation>befaultcontentUnit is concrete implementation of
        abstract conentUnit that also allows attachment of process
        specification to it. Its instace can be used in the instance
        document in the place where contentUnit declared to be present.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base = "contentUnitType">
          <xsd:sequence>
            <xsd:element name = "process">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name = "parameter" minOccurs = "0" maxOccurs = "unbounded">
                    <xsd:complexType>
                      <xsd:simpleContent>
                        <xsd:extension base = "xsd:string">
                          <xsd:attribute name = "name" use = "required" type = "xsd:string"/>
                        </xsd:extension>
                      </xsd:simpleContent>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:sequence>
                <xsd:attribute name = "textInfo" type = "xsd:string"/>
                <xsd:attribute name = "behaviorID" type = "xsd:string"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>
```

7.3.1.2 Sample instance of behavior content unit

```
<ipMapSec>
  < behavioralContentUnit id="productFile">
    <textinfo>This file contains the result of the YYY experience relative to the Earth Exploration Mission of May 1912...</textinfo>
    <dataptr>ExperienceFile</dataptr>
    <process textinfo="Process EAST CHECK" behaviorID="EAST">
      <!-- call the corresponding behavior -->
      <param name="EASTFile">eastDDR</param>
      <!-- no value for param means local contentunit -->
      <param name="DATAFIle"/>
    </process>
  </ContentUnit>
</ipMapSec>
```

7.4 SEMANTICS AND ISSUE

7.4.1 ISSUES

1. Content unit as the result of a product
2. Attributes vs elements in content unit

7.4.2 DATA UNIT TYPES(PROPOSED)

7.4.2.1 Exchange Data Unit (EDU)

The Exchange Data Unit is a set of objects that has been packaged for a reason. It is assumed that all the objects are related though the exact relationship is not one of the predefined types defined by the units below

7.4.2.2 Application Data Unit (ADU)

An Application Data Unit is a package type where all the objects are related to one or more primary content objects of interest

7.4.2.3 Description Data Unit (DDU)

A Description Data Unit is a package type where all the content object are metadata objects intended to update the metadata repository of the consume

7.4.2.4 Process Description Unit (PDU)

A Process Description Unit can range from an automated scripting language to an English language description of the steps a person /intelligent agent. The formal languages/schema used to define the PDU will be part of a future version

7.4.2.5 Software Intallation Unit

Based on Grid P an SIU holds all of the information that the XFDU processor need in order to create a software instance on a particular system

8 Data Object Entry Section

8.1 OVERVIEW

An Data Object Entry Section contains Data Object Entries that contain current data object content and any required data to restore them to the form intended for the original designated community. It has two possible subsidiary elements, The dataObject element provides access to the current content files for a XFDU document. The infoObjEntry must contain exactly 1 data object element that may contain 0 or more FLocat elements, which provides pointer(s) to content file(s), and/or an FContent element, which wraps an encoded version of the file. The infoObjEntry may contain one or more transformation elements that contain all of the information required to reverse each transformation applied to the data object and return the original binary data object.

8.2 XML SCHEMA

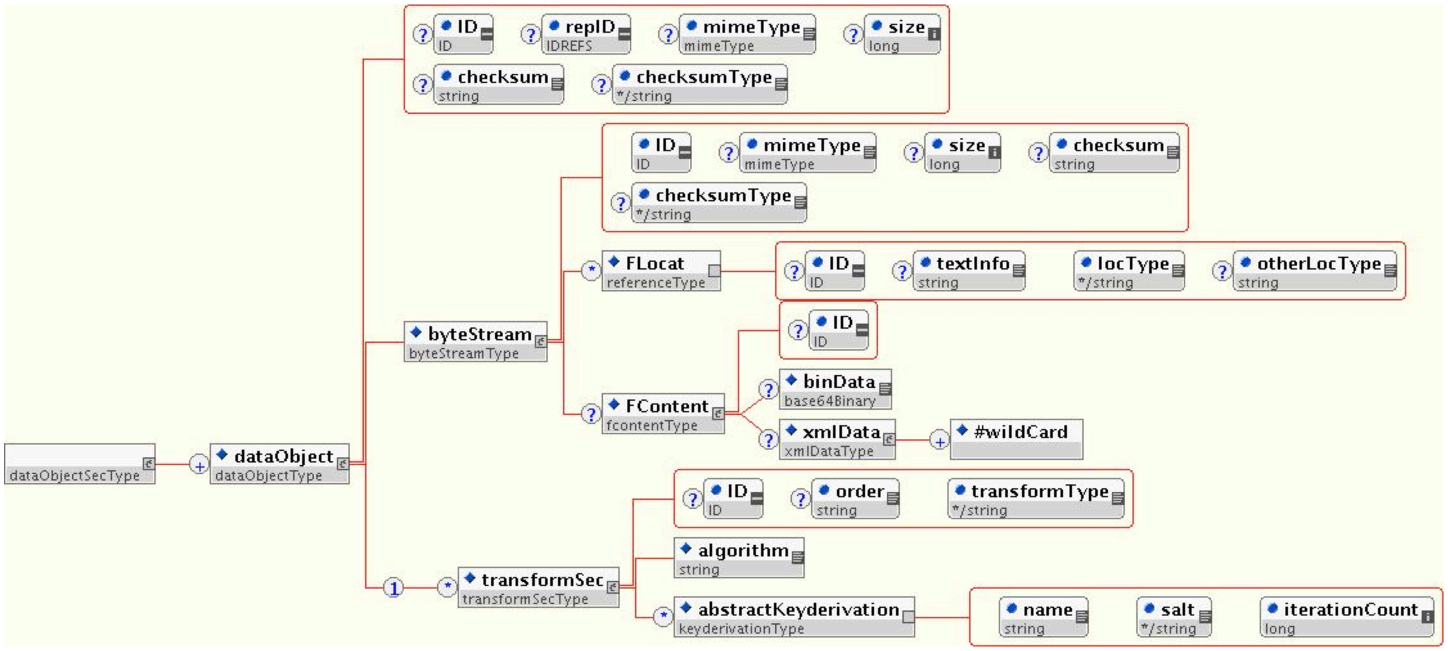


Figure 11

```
<xsd:complexType name = "keyderivationType">
<xsd:annotation>
  <xsd:documentation>key derivation type contains the information
  that was used to derive the encryption key for this dataObject.
  Key derivation type contains:
   name - name of algorithm used
  salt - 16-byte random seed used for that algorithm initialization
```

```
    iterationCount - number of iterations used by the algorithm to derive the key
    </xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name = "name" use = "required" type = "xsd:string"/>
  <xsd:attribute name = "salt" use = "required">
    <xsd:simpleType>
      <xsd:restriction base = "xsd:string">
        <xsd:length value = "16"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name = "iterationCount" use = "required" type = "xsd:long"/>
</xsd:complexType>

<xsd:element name = "abstractKeyderivation" type = "keyderivationType" abstract = "true">
  <xsd:annotation>
    <xsd:documentation>
    abstractKeyderivation is declared abstract
    so that it can be used for element substitution in cases when default key derivation is not
    sufficient. In order for creating more specific key derivation constructs, one would have to
    extend from keyderivationType to a concrete type, and then create an element of that new type. Finally,
    in an instance of XML governed by this schema, the reference to key derivation in an instance of
    transformSec element would point not to instance of keyderivation element, but rather instance of the
    custom element. In other words, keyderivation would be SUBSTITUTED with a concrete key derivation element.
    In cases where default functionality is sufficient, the provided defaultKeyderivation element can be used for the
    substitution.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:element name = "keyderivation" type = "keyderivationType" substitutionGroup = "abstractKeyderivation">
  <xsd:annotation>
    <xsd:documentation>
      Default implementation of key derivation type.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:complexType name = "transformSecType">
  <xsd:annotation>
    <xsd:documentation>transformSecType: transformation information An element
    of transformsecType contains all of the information required to reverse the
    transformations applied to the original contents of the dataObject. It
    has two possible subsidiary elements: The algorithm element
    contains information about the algorithm used to encrypt the
    data. The key-derivation element contains the information that
    was used to derive the encryption key for this dataObject It has three
    attributes: 1. ID: an XML ID 2. transoformType: one of n predefined
    transformations types. Current valid types are compression,
    encryption, authentication. 3. order: If there are more than one
    transformation elements in an infoObjEntry this integer indicates
    the order in which the reversal transformations should be applied.


    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name = "algorithm" type = "xsd:string">
```

```
      <xsd:annotation>
        <xsd:documentation>algorithm element contains information
          about the algorithm used to encrypt the data.


        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element ref = "abstractKeyderivation" minOccurs = "0" maxOccurs = "unbounded"/>
  </xsd:sequence>
  <xsd:attribute name = "ID" type = "xsd:ID"/>
  <xsd:attribute name = "order" type = "xsd:string"/>
  <xsd:attribute name = "transformType" use = "required">
    <xsd:simpleType>
      <xsd:restriction base = "xsd:string">
        <xsd:enumeration value = "COMPRESSION"/>
        <xsd:enumeration value = "AUTHENTICATION"/>
        <xsd:enumeration value = "ENCRYPTION"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>>
</xsd:element>
<xsd:element name = "FContent">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base = "fcontentType"/>
    </xsd:complexContent>
  </xsd:complexType>
<xsd:complexType name = "byteStreamType">
  <xsd:annotation>
    <xsd:documentation>byteStreamType: An element of  byteStreamType
    provides access to the current content of dataObjects for a XFDU
    document. The byteStreamType: has the following four attributes: ID (an XML ID);
    mimeType: the MIME type for the dataObject; size: the size of the dataObject
    in bytes; checksum: a checksum for dataObject; checksumType: type of checksum
    algorithms used to compute checksum. The data contained in these attributes is relevant to
    final state of data object after all possible transformations of the original data.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name = "FLocat" type = "referenceType" minOccurs = "0" maxOccurs = "unbounded"/>
    <xsd:element name = "FContent" type = "fcontentType" minOccurs = "0"/>
  </xsd:sequence>
  <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
  <xsd:attribute name = "mimeType" type = "mimeType"/>
  <xsd:attribute name = "size" type = "xsd:long"/>
  <xsd:attribute name = "checksum" type = "xsd:string"/>
  <xsd:attribute name = "checksumType">
    <xsd:simpleType>
      <xsd:restriction base = "xsd:string">
        <xsd:enumeration value = "HAVAL"/>
        <xsd:enumeration value = "MD5"/>
        <xsd:enumeration value = "SHA-1"/>
        <xsd:enumeration value = "SHA-256"/>
        <xsd:enumeration value = "SHA-384"/>
        <xsd:enumeration value = "SHA-512"/>
        <xsd:enumeration value = "TIGER"/>
        <xsd:enumeration value = "WHIRLPOOL"/>
        <xsd:enumeration value = "CRC32"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name = "dataObjectType">
  <xsd:annotation>
    <xsd:documentation>dataObjectType : An element of dataObjectType
    contains current byteStream content and any required data to restore
    them to the form intended for the original designated community.
    It has two possible subsidiary elements: The byteStream element
    provides access to the current content dataObjects for an XFDU
    document. An element of dataObjectType must contain exactly 1 byteStream element
    that may contain an FLocat element, which provides a pointer to
    a content byteStream, and/or an FContent element, which wraps an
    encoded version of the dataObject. An element of dataObjectType may contain one or
    more transformation elements that contain all of the
    information required to reverse each transformation applied to
    the dataObject and return the original binary data object.
    The infoObjEntry has the following five attributes: 1. ID: an XML ID
    2, mimeType: the MIME type for the dataObject 3. size: the size of the dataObject
    in bytes 4. checksum: a checksum for dataObject 5. checksumType: type of checksum algorithms used to
    compute checksum   6 repID  list of representation metadata IDREFs. The size, checksum, checksumtype and
    mime type are related to the original data before any transformations occured.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name = "byteStream" type = "byteStreamType"/>
    <xsd:sequence>
      <xsd:element name = "transformSec" type = "transformSecType" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:sequence>
  <xsd:attribute name = "ID" type = "xsd:ID"/>
  <xsd:attribute name = "repID" type = "xsd:IDREFS"/>
  <xsd:attribute name = "mimeType" type = "mimeType"/>
  <xsd:attribute name = "size" type = "xsd:long"/>
  <xsd:attribute name = "checksum" type = "xsd:string"/>
  <xsd:attribute name = "checksumType">
    <xsd:simpleType>
      <xsd:restriction base = "xsd:string">
        <xsd:enumeration value = "HAVAL"/>
        <xsd:enumeration value = "MD5"/>
        <xsd:enumeration value = "SHA-1"/>
        <xsd:enumeration value = "SHA-256"/>
        <xsd:enumeration value = "SHA-384"/>
        <xsd:enumeration value = "SHA-512"/>
        <xsd:enumeration value = "TIGER"/>
        <xsd:enumeration value = "WHIRLPOOL"/>
        <xsd:enumeration value = "CRC32"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
```

```
  <xsd:complexType name = "dataObjectSecType">
    <xsd:annotation>
      <xsd:documentation>dataObjectSecType : a container for one or more elements of dataObjectType
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name = "dataObject" type = "dataObjectType" maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
```

8.3 EXAMPLES

8.3.1 VERIFY THE CHECKSUM OF THE FILE

Reader of the document can verify checksum of animation file by comparing its checksum with checksum value specified in checksum attribute of dataObject element.

```
  <dataObjectSec>
    <dataObject repID = "mathMLAlgRepMD" ID = "mpeg21">
      <byteStream mimeType = "video/mpeg" checksum = "b3eb4b34" ID = "mpeg21AnimData" checksumType =
"CRC32" size = "414131">
        <FLocat locType = "URL" ns2:href = "file:packagesamples/scenario1/mpeg21.mpg" xmlns:ns2 =
"http://www.w3.org/TR/xlink"/>
      </byteStream>
    </dataObject>
```

8.3.2 SPECIFICATION OF MIMETYPE AND CHECKSUM WITH TRANSFORMATIONS

Mimetype and checksum are specified at two levels. infoObjEntry's values of mimeType and checksum attributes specified mime type and checksum of the original data object, that is before any transformations happened. dataObject's mimeType and checksum attributes specifies mimeType and checksum of the dataObject after all transformations happened on it (e.g. encryption).

```
    <dataObject size = "0" checksumType = "CRC32" checksum = "6d0e30ea" mimeType = "application/pdf" ID =
"ATDMD">
      <byteStream mimeType = "application/octetstream" checksum = "ad78ad5d" ID = "atdMDbs" checksumType =
"CRC32" size = "110874">
        <FLocat locType = "URL" ns3:href = "file:packagesamples/scenario1/atd.pdf" xmlns:ns3 =
"http://www.w3.org/TR/xlink"/>
      </byteStream>
      <transformSec transformType = "ENCRYPTION">
        <algorithm>blowfish</algorithm>
      </transformSec>
    </dataObject>
```

8.3.3 REFERENCING AND INCLUSION OF DATA CONTENT

The data that existed at the moment of packaging base64 encoded within binData. Also, FLocat element points to HTTP GET URL where the latest version of data can be obtained.

```
    <dataObject mimeType = "application/octetstream" ID = "orbitalData">
      <byteStream checksum = "b3eb4b34" ID = "orbitData" checksumType = "CRC32">
        <FLocat locType = "URL" ns7:href = "http://coin.gsfc.nasa.gov:8080/ims-bin/3.0.1/nph-
ims.cgi?msubmit=yes&amp;lastmode=SRCHFORM" xmlns:ns7 = "http://www.w3.org/TR/xlink"/>
        <FContent>

  <binData>UEsDBBQACAAIAKqMBC8AAAAAAAAAAAAAAAAPAAAAeGZkdS8uY2xhc3NwYXRotZXfS8MwEMff/StK35
OugqCwH4hO0I...</binData>
        </FContent>
      </byteStream>
  </dataObject>
```

8.4 SEMANTICS AND ISSUES

These are proposed sematics for the existence of most than one Flocat or FContent element in a dataObject

1. One fContent and 1 fLocat means the fContent should serve as backup if the fLocat is not accessable
2. One Fllocat referencing a object in the XFDU and One fLocat accessing an Object outside the XFDU means the object located within the package should serve as backup if the remote fLocat is not available
3. Multiple fLocats indicate the referenced objects must be concatenated be the content unit metadata is correct

9 Schema for MdScctype

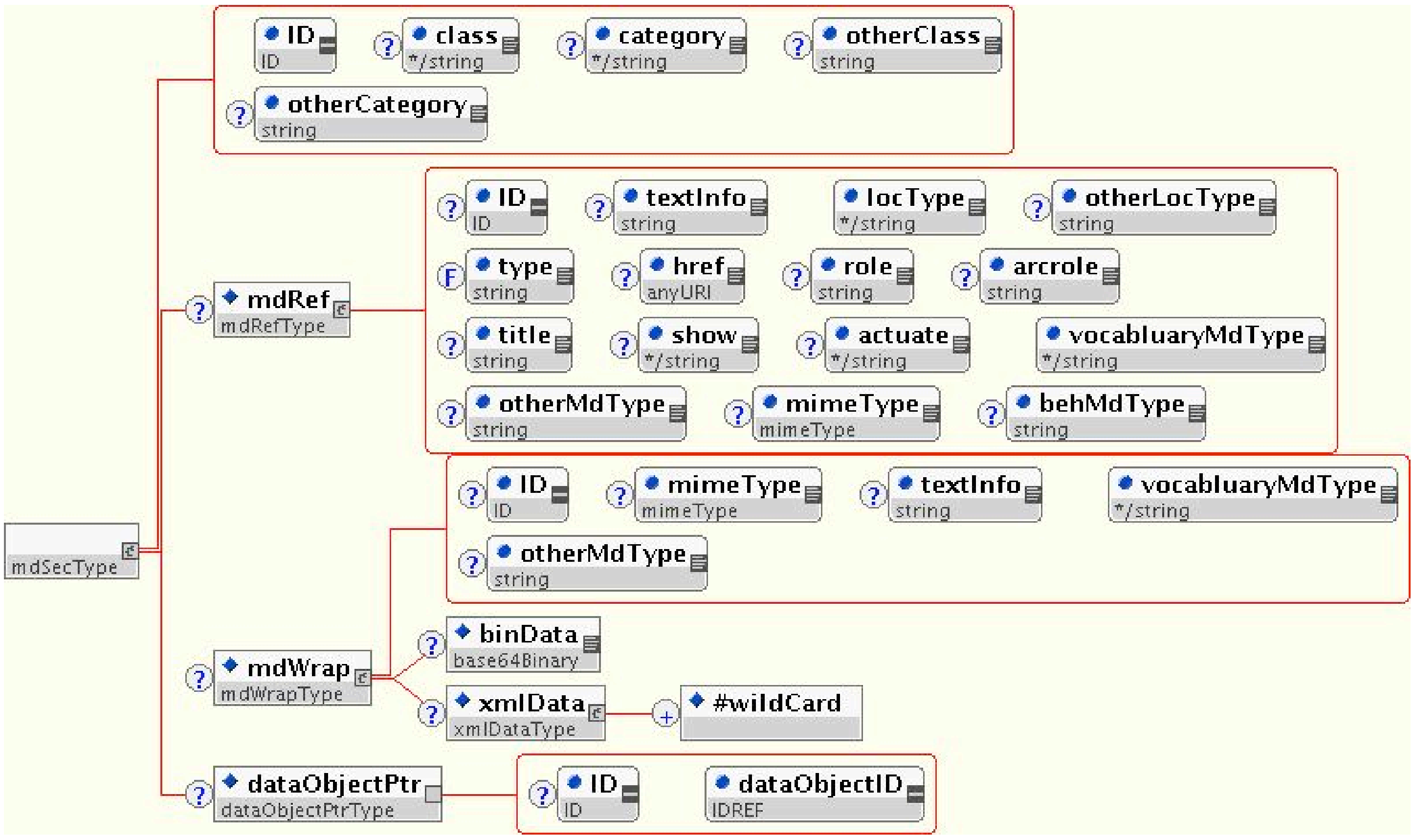


Figure 3

```
<xsd:complexType name = "mdSecType">
  <xsd:annotation>
    <xsd:documentation>mdSecType (metadata section) Complex Type A generic
    framework for pointing to/including metadata within a XFDU
    document, a la Warwick Framework. An mdSec element may have the
    following attributes:
    1. ID: an XML ID for this element.
    2. class - concrete type of metadata represented by this element of mdSecType
    3. category - type of metadata class to which this metadata belongs (e.g. DMD.REP, etc.)


    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name = "mdRef" type = "mdRefType" minOccurs = "0"/>
    <xsd:element name = "mdWrap" type = "mdWrapType" minOccurs = "0"/>
    <xsd:element name = "dataObjectPtr" type = "dataObjectPtrType" minOccurs = "0"/>
  </xsd:sequence>
  <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
  <xsd:attribute name = "class">
    <xsd:simpleType>
      <xsd:restriction base = "xsd:string">
        <xsd:enumeration value = "DED"/>
        <xsd:enumeration value = "SYNTAX"/>
        <xsd:enumeration value = "FIXITY"/>
        <xsd:enumeration value = "PROVENANCE"/>
        <xsd:enumeration value = "CONTEXT"/>
        <xsd:enumeration value = "REFERENCE"/>
        <xsd:enumeration value = "DESCRIPTION"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name = "category">
    <xsd:simpleType>
      <xsd:restriction base = "xsd:string">
        <xsd:enumeration value = "REP"/>
        <xsd:enumeration value = "PDI"/>
        <xsd:enumeration value = "DMD"/>
        <xsd:enumeration value = "OTHER"/>
        <xsd:enumeration value = "ANY"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name = "otherClass" type = "xsd:string"/>
  <xsd:attribute name = "otherCategory" type = "xsd:string"/>
</xsd:complexType>
```

10 Metadata Section Type and Metadata Objects

10.1 OVERVIEW

This section illustrated the instantiation of mdSec into meta data objects and the metadataSec type as a container object for the

10.2 XML SCHEMA

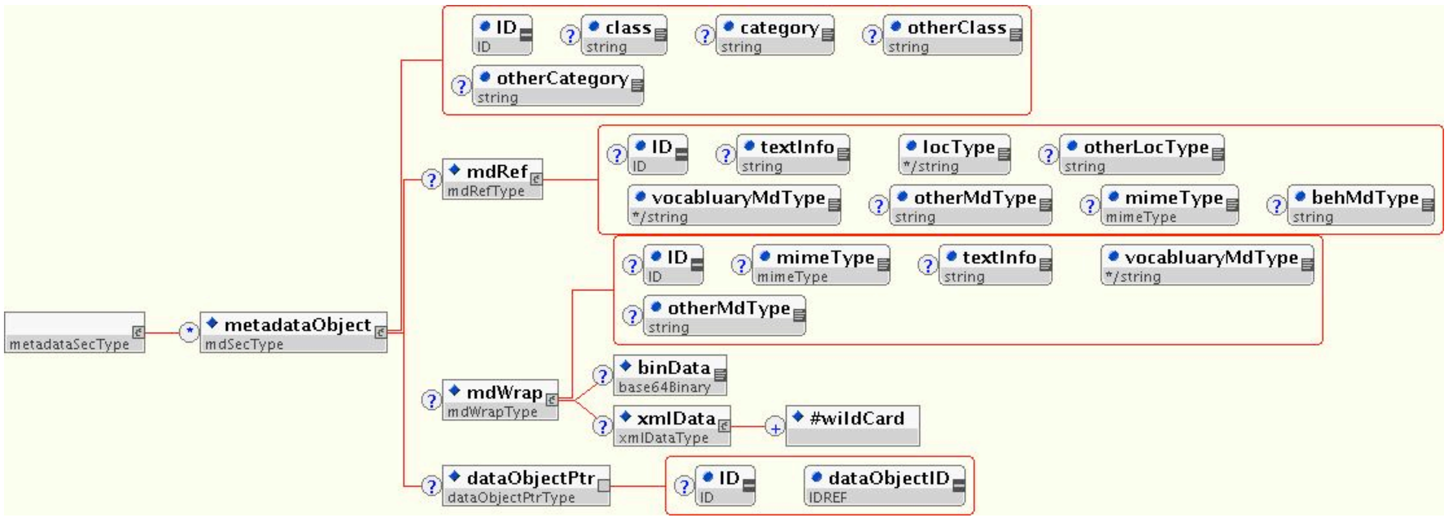


Figure 4

```
xsd:complexType name = "metadataSecType">
    <xsd:sequence>
      <xsd:element name = "metadataObject" type = "mdSecType" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
```

11 Behavior Section

11.1 OVERVIEW

A **behavior section** can be used to associate executable behaviors with content in the XFDU object. A behavior section has an interface definition element that represents an abstract definition of the set of behaviors represented by a particular behavior section. A behavior section also has an behavior mechanism which is a module of executable code that implements and runs the behaviors defined abstractly by the interface definition. An behavior section may contain the following elements:

Interface Definition Element

The **interface definition element** contains a pointer an abstract definition of a set of related behaviors. These abstract behaviors can be associated with the content of a XFDU object. The interface definition element will be a pointer to another object (an interface definition object). An interface definitio object could be another XFDU object, or some other entity (e.g. a WSDL file). Ideally, an interface definition object should contain metadata that describes a set of behaviors or methods, may also contain files that describe the intended usage of the behaviors, and possibly files that represent different expressions of the interface definition. The interfaceDef element is optional to allow for cases where an interface definition can be obtained from a behavior mechanism object (see the mechanism element of the behaviorSec). InterfaceDef extends from objectType and adds ability of specifying inputParameter that can be either just a string value or pointer to the content in this package

Mechanism:

A mechanism element contains a pointer to an executable code module that implements a set of behaviors defined by an interface definition. The mechanism element will be a pointer to another object (a mechanism object). A mechanism object could be anotherXFDU object, or some other entity (e.g., a WSDL file). A mechanism object should contain executable code, pointers to executable code, or specifications for binding to network services (e.g., web services).

11.2 XML SCHEMA

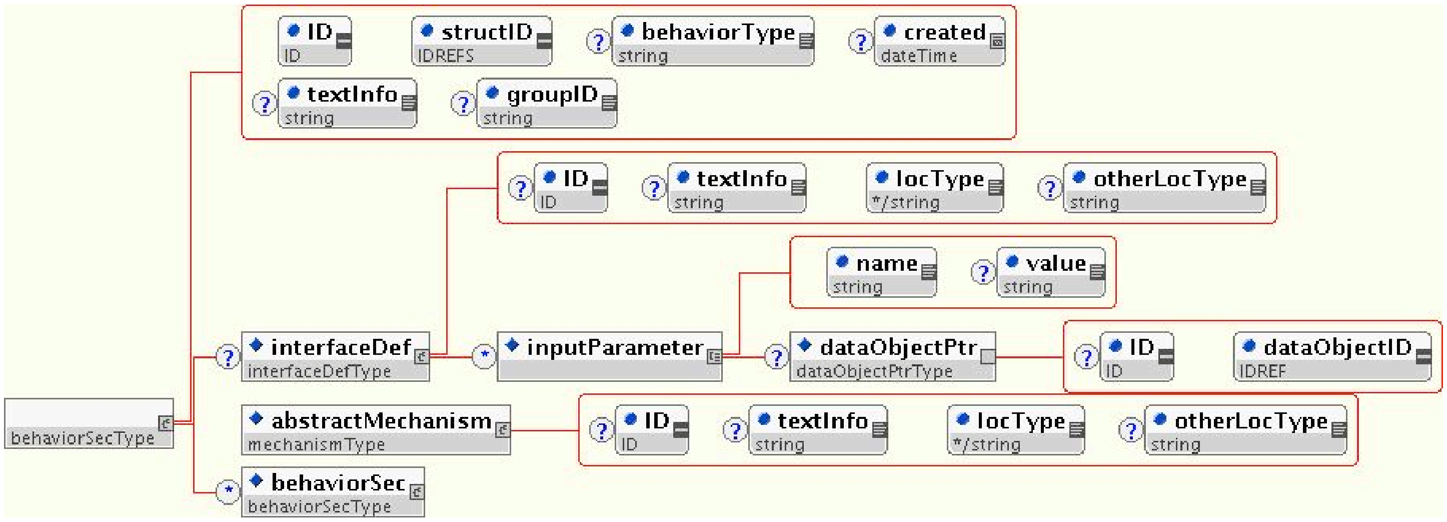


Figure 5

```
<xsd:complexType name = "interfaceDefType">
    <xsd:annotation>
      <xsd:documentation>interfaceDefType: interface definition object. The
        interface definition type contains a pointer an abstract
        definition of a set of related behaviors. These abstract
        behaviors can be associated with the content of a XFDU object.
        The interface definition element will be a pointer to another
        object (an interface definition object). An interface definition
        object could be another XFDU object, or some other entity (e.g.,
        a WSDL source). Ideally, an interface definition object should
        contain metadata that describes a set of behaviors or methods.
        It may also contain files that describe the intended usage of
        the behaviors, and possibly files that represent different
        expressions of the interface definition. An element of interfaceDefType
        is optional to allow for cases where an interface
        definition can be obtained from a behavior mechanism object (see
        the mechanism element of the behaviorSec).
        interfaceDef extends from referenceType and adds ability of specifying inputParameter
        that can be either just a string value or pointer to the content in this package


      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base = "referenceType">
        <xsd:sequence>
          <xsd:element name = "inputParameter" minOccurs = "0" maxOccurs = "unbounded">
            <xsd:complexType mixed = "true">
              <xsd:sequence>
                <xsd:element name = "dataObjectPtr" type = "dataObjectPtrType" minOccurs = "0"/>
              </xsd:sequence>
              <xsd:attribute name = "name" use = "required" type = "xsd:string"/>
              <xsd:attribute name = "value" type = "xsd:string"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name = "behaviorSecType">
    <xsd:annotation>
      <xsd:documentation>behaviorSecType: Complex Type for Behaviors. A
        behavior section can be used to associate executable behaviors
        with content in the XFDU object. A behavior section has an
        interface definition element that represents an abstract
        definition of the set of behaviors represented by a particular
        behavior section. A behavior section also has an behavior
        mechanism which is a module of executable code that implements
        and runs the behaviors defined abstractly by the interface
        definition. An behavior section may have the following
        attributes: 1. ID: an XML ID for the element 2. structID: IDREFS
        to structMap sections or divs within a structMap in the XFDU
        document. The content that the structID attribute points to is
        considered "input" to the behavior mechanism (executable)
        defined for the behaviorSec. 3. behaviorType: a behavior type
        identifier for a given set of related behaviors. 4. created:
        date this behavior section of the XFDU object was created. 5.
        textInfo: a description of the type of behaviors this section
        represents. 6. groupID: an identifier that establishes a
        correspondence between this behavior section and behavior
        sections. Typically, this will be used to facilitate versioning
        of behavior sections.  behavior  section may also include another behavior section for chaining of behaviors
        Concrete implementation of mechanism (wsdlMechanism,antMechanism,javaMechanism, etc) have to be used in
the instance document.


      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name = "interfaceDef" type = "interfaceDefType" minOccurs = "0"/>
      <xsd:element ref = "abstractMechanism"/>
      <xsd:element name = "behaviorSec" type = "behaviorSecType" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
    <xsd:attribute name = "structID" use = "required" type = "xsd:IDREFS"/>
    <xsd:attribute name = "behaviorType" type = "xsd:string"/>
    <xsd:attribute name = "created" type = "xsd:dateTime"/>
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
    <xsd:attribute name = "groupID" type = "xsd:string"/>
  </xsd:complexType>
  <xsd:element name = "abstractMechanism" type = "mechanismType" abstract = "true">
    <xsd:annotation>
      <xsd:documentation>abstractMechanism is abstract implementation of
        mechanismType. It cannot be instanciated in the instance
        document. Instead, concrete implementations would have to be
        used which are declared part of mechanism substitutionGroup
      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name = "mechanismType">
```

```
    <xsd:annotation>
      <xsd:documentation>mechanismType: executable mechanism. An element of  mechanismType
        contains a pointer to an executable code module that
        implements a set of behaviors defined by an interface
        definition. The mechanism element will be a pointer to another
        object (a mechanism object). A mechanism object could be another
        XFDU object, or some other entity (e.g., a WSDL source). A
        mechanism object should contain executable code, pointers to
        executable code, or specifications for binding to network
        services (e.g., web services).
        mechanismType is declared as base type for concrete implementations of mechanism
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base = "referenceType"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:complexType name = "behaviorSecType">
    <xsd:annotation>
      <xsd:documentation>behaviorSecType: Complex Type for Behaviors. A
        behavior section can be used to associate executable behaviors
        with content in the XFDU object. A behavior section has an
        interface definition element that represents an abstract
        definition of the set of behaviors represented by a particular
        behavior section. A behavior section also has an behavior
        mechanism which is a module of executable code that implements
        and runs the behaviors defined abstractly by the interface
        definition. An behavior section may have the following
        attributes: 1. ID: an XML ID for the element 2. structID: IDREFS
        to structMap sections or divs within a structMap in the XFDU
        document. The content that the structID attribute points to is
        considered "input" to the behavior mechanism (executable)
        defined for the behaviorSec. 3. behaviorType: a behavior type
        identifier for a given set of related behaviors. 4. created:
        date this behavior section of the XFDU object was created. 5.
        textInfo: a description of the type of behaviors this section
        represents. 6. groupID: an identifier that establishes a
        correspondence between this behavior section and behavior
        sections. Typically, this will be used to facilitate versioning
        of behavior sections.  behavior  section may also include another behavior section for chaining of behaviors
        Concrete implementation of mechanism (wsdlMechanism,antMechanism,javaMechanism, etc) have to be used in
the instance document.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name = "interfaceDef" type = "interfaceDefType" minOccurs = "0"/>
      <xsd:element ref = "abstractMechanism"/>
      <xsd:element name = "behaviorSec" type = "behaviorSecType" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
    <xsd:attribute name = "structID" use = "required" type = "xsd:IDREFS"/>
    <xsd:attribute name = "behaviorType" type = "xsd:string"/>
    <xsd:attribute name = "created" type = "xsd:dateTime"/>
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
    <xsd:attribute name = "groupID" type = "xsd:string"/>
  </xsd:complexType>
```

11.3 EXAMPLES

Abstract element mechanism cannot be instantiated in XFDU XML document. The following examples show how abstract mechanism specified in the schema can be extended with concrete mechanism definitions via substitutionGroup technique. The concrete mechanisms can be then used in XFDU XML documents.

11.3.1 WEB-SERVICE-BASED MECHANISM

wsMechanism element is of type mechanismType and is part of mechanism substitution group. wsMechanism defines web-service-based mechanism. This means that access point or WSDL document of the web service that implements this mechanims should be specified via linking. This mechanism can used as a concrete mechanism inside of the behaviorSec.

```
  <xsd:element name = "wsMechanism" type = "mechanismType" substitutionGroup = "abstractMechanism">
    <xsd:annotation>
      <xsd:documentation>wsMechanism is concrete implementation of
        abstract mechanism which implements mechanism based on Web
        service. Its instace can be used in the instance document in the
        place where mechanism declared to be present.
      </xsd:documentation>
    </xsd:annotation>
      </xsd:element>
```

11.3.2 .JAVA-BASED MECHANISM

javaMechanism element is of type mechanismType and is part of mechanism substitution group. javaMechanism defines java-based mechanism. This meachanism would specify location of JAR file where executable JAVA code resided and fully qualified name of the main class. This mechanism can used as a concrete mechanism inside of the behaviorSec.

```
<xsd:element name = "javaMechanism" substitutionGroup = "abstractMechanism">
    <xsd:annotation>
      <xsd:documentation>javaMechanism is concrete implementation of
        abstract mechanism which implements Java-based mechanism.
        xlink:href element points to fully qualified name of the Java
        class that implements the mechanism. FLocate element specifies
        location of the jar file where that class is packaged. File can
        be local to this XFDU package, or located on the remote server,
        or somewhere on the local file system.An instace of
        javaMechanism can be used in the instance document in the place
        where mechanism declared to be present.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base = "mechanismType">
          <xsd:sequence>
            <xsd:element name = "FLocat" type = "referenceType"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
```

```
  </xsd:complexType>
    </xsd:element>
```

11.3.3 ANT BASED MECHANISM

antMechanism element is of type mechanismType and is part of mechanism substitution group. antMechanism defines mechanism that is based on Apache Foundation’s ANT tool. One or several ANT-specific XML scripts can be included via xmlData element. At runtime this scripts would have to be interpreted by ANT to execute appropriate software. Software can be included either in the package or reside on the file system. It can be portable, like JAVA code, or operating system specific. This mechanism can used as a concrete mechanism inside of the behaviorSec.

```
<xsd:element name = "antMechanism" substitutionGroup = "abstractMechanism">
  <xsd:annotation>
    <xsd:documentation>antMechanism is concrete implementation of
      abstract mechanism which implements ANT-based mechanism.
      xlink:href element points to a location of ANT script to be
      executed. Also, xmlData element can contain ANT specific XML. An
      instace of antMechanism can be used in the instance document in
      the place where mechanism declared to be present
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base = "mechanismType">
        <xsd:sequence>
          <xsd:element name = "xmlData" type = "xmlDataType" minOccurs = "0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

11.4SEMATICS AND ISSUES

12 Full XML Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by Turbo XML 2.3.1.100. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<xsd:schema xmlns:xlink = "http://www.w3.org/TR/xlink" xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
   elementFormDefault = "qualified">
  <xsd:import namespace = "http://www.w3.org/TR/xlink" schemaLocation =
"http://www.loc.gov/standards/mets/xlink.xsd"/>
  <xsd:attributeGroup name = "LOCATION">
    <xsd:annotation>
      <xsd:documentation>
        This attribute group aggregates attributes that can be used for specifiying type of location
        This group includes following attributes:
        locType specifies location type (e.g. URN,URL)
        otherLocType specifies location type in case locType has value of OTHER


      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name = "locType" use = "required">
      <xsd:simpleType>
        <xsd:restriction base = "xsd:string">
          <xsd:enumeration value = "URN"/>
          <xsd:enumeration value = "URL"/>
          <xsd:enumeration value = "PURL"/>
          <xsd:enumeration value = "HANDLE"/>
          <xsd:enumeration value = "DOI"/>
          <xsd:enumeration value = "OTHER"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name = "otherLocType" type = "xsd:string"/>
  </xsd:attributeGroup>
  <xsd:attributeGroup name = "METADATA">
    <xsd:annotation>
      <xsd:documentation>
        This attribute group aggregates attributes that can be used for specifying metadata type
        This group includes following attributes:
        vocabluaryMdType specifies location type (e.g. MARC.DDI)
        otherMdType specifies location type in case mdType has value of OTHER


      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name = "vocabluaryMdType" use = "required">
      <xsd:simpleType>
        <xsd:restriction base = "xsd:string">
          <xsd:enumeration value = "MARC"/>
          <xsd:enumeration value = "EAD"/>
          <xsd:enumeration value = "DC"/>
          <xsd:enumeration value = "NISOIMG"/>
          <xsd:enumeration value = "LC-AV"/>
          <xsd:enumeration value = "VRA"/>
          <xsd:enumeration value = "TEIHDR"/>
          <xsd:enumeration value = "DDI"/>
          <xsd:enumeration value = "FGDC"/>
          <xsd:enumeration value = "EAST"/>
          <xsd:enumeration value = "OTHER"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name = "otherMdType" type = "xsd:string"/>
  </xsd:attributeGroup>
  <xsd:simpleType name = "mimeType">
    <xsd:restriction base = "xsd:string">
      <xsd:enumeration value = "image/gif"/>
      <xsd:enumeration value = "image/jpeg"/>
      <xsd:enumeration value = "image/jpeg2000"/>
      <xsd:enumeration value = "image/png"/>
      <xsd:enumeration value = "image/tiff"/>
      <xsd:enumeration value = "image/geo-tiff"/>
      <xsd:enumeration value = "application/octetstream"/>
      <xsd:enumeration value = "application/pdf"/>
      <xsd:enumeration value = "application/x-hdf"/>
      <xsd:enumeration value = "application/hdf-eos"/>
      <xsd:enumeration value = "application/fits"/>
      <xsd:enumeration value = "application/eas"/>
      <xsd:enumeration value = "application/pds"/>
      <xsd:enumeration value = "application/xfdu"/>
      <xsd:enumeration value = "application/cdf"/>
      <xsd:enumeration value = "application/net-cdf"/>
      <xsd:enumeration value = "text/xml"/>
      <xsd:enumeration value = "text/html"/>
      <xsd:enumeration value = "text/plain"/>
      <xsd:enumeration value = "application/doc"/>
      <xsd:enumeration value = "application/rtf"/>
      <xsd:enumeration value = "xml/ded"/>
      <xsd:enumeration value = "xml/votable"/>
      <xsd:enumeration value = "video/mpeg"/>
      <xsd:enumeration value = "video/mpeg-7"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:attribute name = "mustUnderstand" type = "xsd:boolean"/>
  <xsd:attribute name = "namespace" type = "xsd:string"/>
  <xsd:complexType name = "referenceType">
    <xsd:attribute name = "ID" type = "xsd:ID"/>
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
    <xsd:attributeGroup ref = "LOCATION"/>
    <xsd:attributeGroup ref = "xlink:simpleLink"/>
  </xsd:complexType>
  <xsd:complexType name = "mdSecType">
    <xsd:annotation>
      <xsd:documentation>mdSecType (metadata section) Complex Type A generic
      framework for pointing to/including metadata within a XFDU
      document, a la Warwick Framework. An mdSec element may have the
      following attributes:
      1. ID: an XML ID for this element.
      2. class - concrete type of metadata represented by this element of mdSecType
      3. category - type of metadata class to which this metadata belongs (e.g. DMD.REP, etc.)
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name = "mdRef" type = "mdRefType" minOccurs = "0"/>
      <xsd:element name = "mdWrap" type = "mdWrapType" minOccurs = "0"/>
      <xsd:element name = "dataObjectPtr" type = "dataObjectPtrType" minOccurs = "0"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
    <xsd:attribute name = "class">
      <xsd:simpleType>
        <xsd:restriction base = "xsd:string">
          <xsd:enumeration value = "DED"/>
          <xsd:enumeration value = "SYNTAX"/>
          <xsd:enumeration value = "FIXITY"/>
          <xsd:enumeration value = "PROVENANCE"/>
          <xsd:enumeration value = "CONTEXT"/>
          <xsd:enumeration value = "REFERENCE"/>
          <xsd:enumeration value = "DESCRIPTION"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name = "category">
      <xsd:simpleType>
        <xsd:restriction base = "xsd:string">
          <xsd:enumeration value = "REP"/>
          <xsd:enumeration value = "PDI"/>
          <xsd:enumeration value = "DMD"/>
          <xsd:enumeration value = "OTHER"/>
          <xsd:enumeration value = "ANY"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name = "otherClass" type = "xsd:string"/>
    <xsd:attribute name = "otherCategory" type = "xsd:string"/>
  </xsd:complexType>
  <xsd:complexType name = "packHdrType">
    <xsd:annotation>
      <xsd:documentation>packHdrType: Complex Type for metadata about the
      mapping of the logical packages to the physical structures. The
      package header section consists of three possible subsidiary
      sections: envirMD (specification of the hardware and software
      platform which created this package), behaviorMD (behavior mechanism
      related metadata), and transformMD (the names, classifications, parameter
      names/types and any other information needed to reverse
      transformations used in the XFDU). Both transformMD and behaviorMD have an optional
      mustUnderstand attribute that declares if the reader of this package must
      understand described transformation, behavior mechanisms in order to
      process content of the package. packHdrType has a single
      attribute, ID: an XML ID.


      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name = "envirMD" minOccurs = "0" maxOccurs = "unbounded">
        <xsd:annotation>
```

```
        <xsd:documentation>envirMD: technical metadata. The envirMD element
        provides a wrapper around a generic metadata section that
        should contain technical metadata regarding a dataObject or dataObjects. It
        has a single attribute, ID, which dataObject/dataObjectGrp elements can use
        to reference the technical metadata that applies to them.


        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base = "xsd:string">
            <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name = "behaviorMD" minOccurs = "0" maxOccurs = "unbounded">
      <xsd:annotation>
        <xsd:documentation>
          behaviorMD contains:
          mustUnderstand - indicates if this mechanism must be understood by processor
          description - general description
          mechanismType - type of behavior mechanism (e.g. WS,ANT,JAVA) (should be made into enumeration most
likely)
          namespace - namespace of the specified technology if any
          behaviorMD has an optional xmlData element to include any additional metadata if needed


        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name = "xmlData" type = "xmlDataType" minOccurs = "0" maxOccurs = "unbounded"/>
        </xsd:sequence>
        <xsd:attribute ref = "mustUnderstand"/>
        <xsd:attribute name = "description" type = "xsd:string"/>
        <xsd:attribute name = "mechanismType" type = "xsd:string"/>
        <xsd:attribute ref = "namespace"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name = "transformMD" minOccurs = "0" maxOccurs = "unbounded">
      <xsd:annotation>
        <xsd:documentation>transformMD (the names, classifications, parameter
        names/types and any other information needed to reverse
        transformations used in the XFDU)
        transformMD contains:
        mustUnderstand - indicates if this transformation technology must be understood by processor
        description - gneral description
        algorithmName -name of transformation algorithm
        namespace - namespace of the specified technology if any
        transformMD has optional xmlData element to include any additional metadata if needed


        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name = "xmlData" type = "xmlDataType" minOccurs = "0" maxOccurs = "unbounded"/>
        </xsd:sequence>
        <xsd:attribute name = "description" type = "xsd:string"/>
        <xsd:attribute name = "algorithmName" type = "xsd:string"/>
        <xsd:attribute ref = "mustUnderstand"/>
        <xsd:attribute ref = "namespace"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name = "ID" type = "xsd:ID"/>
</xsd:complexType>
<xsd:complexType name = "mdRefType">
  <xsd:annotation>
    <xsd:documentation>mdRefType: metadata reference. An element of mdRefType is a
    generic element used throughout the XFDU schema to provide a
    pointer to metadata which resides outside the XFDU document. mdRefType
    has the following attributes: 1. ID: an XML ID; 2. locType: the
    type of locator contained in the body of the element; 3.
    otherLocType: a string indicating an alternative locType when
    the locType attribute value is set to "OTHER."; 4. xlink:href:
    see XLink standard (http://www.w3.org/TR/xlink) 5. xlink:role:
    "" 6. xlink:arcrole: "" 7. xlink:title: "" 8. xlink:show: "" 9.
    xlink:actuate: "" 10. mimeType: the MIME type for the metadata
    being pointed at; 11. MDType: the type of metadata being pointed
    at (e.g., MARC, EAD, etc.); 12. behMdType: a string indicating
    an alternative mdType when the mdType attribute value is set to
    "OTHER."; 13. textInfo: a label to display to the viewer of the
    XFDU document identifying the metadata; and NB: mdRef is an empty element. The location of the
    metadata must be recorded in the xlink:href attribute,
    supplemented by the XPTR attribute as needed.


    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base = "referenceType">
      <xsd:attributeGroup ref = "METADATA"/>
      <xsd:attribute name = "mimeType" type = "mimeType"/>
      <xsd:attribute name = "behMdType" type = "xsd:string"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name = "xmlDataType">
  <xsd:annotation>
    <xsd:documentation>A wrapper to contain arbitrary XML content.</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:any namespace = "##any" processContents = "strict" maxOccurs = "unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name = "fcontentType">
  <xsd:annotation>
    <xsd:documentation>
      fContentType encapsulates and agregates a type that can have a choice of either
      binary or xml data


    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice>
    <xsd:element name = "binData" type = "xsd:base64Binary" minOccurs = "0">
      <xsd:annotation>
        <xsd:documentation>A wrapper to contain Base64 encoded metadata.</xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name = "xmlData" type = "xmlDataType" minOccurs = "0"/>
  </xsd:choice>
  <xsd:attribute name = "ID" type = "xsd:ID"/>
</xsd:complexType>
<xsd:complexType name = "mdWrapType">
  <xsd:annotation>
    <xsd:documentation>mdWrapType: metadata wrapper. An element of mdWrapType is a
    generic element used throughout the XFDU schema to allow the
    encoder to place arbitrary metadata conforming to other
    standards/schema within a XFDU document.  The mdWrapType
    can have the following attributes: 1. ID: an XML ID for
    this element; 2. mimeType: the MIME type for the metadata
    contained in the element; 3. MDType: the type of metadata
    contained (e.g., MARC, EAD, etc.); 4. behMdType: a string
    indicating an alternative mdType when the mdType attribute value
    is set to "OTHER."; 5. textInfo: a label to display to the viewer
    of the XFDU document identifying the metadata.


    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base = "fcontentType">
      <xsd:attribute name = "mimeType" type = "mimeType"/>
      <xsd:attribute name = "textInfo" type = "xsd:string"/>
      <xsd:attributeGroup ref = "METADATA"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name = "dataObjectPtrType">
  <xsd:annotation>
    <xsd:documentation>
      The dataObjectPtrType is a type that can be used to refernce dataObjects by dataObjectID.
      The dataObjectPtrType has two attributes:
      1. ID: an XML ID for this element; and
      2. dataObjectID: an IDREF to a dataObject element referenced by contentUnit containing this datatpr.


    </xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name = "ID" type = "xsd:ID"/>
  <xsd:attribute name = "dataObjectID" use = "required" type = "xsd:IDREF"/>
</xsd:complexType>
<xsd:complexType name = "keyderivationType">
  <xsd:annotation>
```

```
      <xsd:documentation>key derivation type contains the information
      that was used to derive the encryption key for this dataObject.
      Key derivation type contains:
       name - name of algorithm used
      salt - 16-byte random seed used for that algorithm initialization
      iterationCount - number of iterations used by the algorithm to derive the key


      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name = "name" use = "required" type = "xsd:string"/>
    <xsd:attribute name = "salt" use = "required">
      <xsd:simpleType>
        <xsd:restriction base = "xsd:string">
          <xsd:length value = "16"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name = "iterationCount" use = "required" type = "xsd:long"/>
  </xsd:complexType>
  <xsd:element name = "abstractKeyderivation" type = "keyderivationType" abstract = "true">
    <xsd:annotation>
      <xsd:documentation>
      abstractKeyderivation is declared abstract
      so that it can be used for element substitution in cases when default key derivation is not
      sufficient. In order for creating more specific key derivation constructs, one would have to
      extend from keyderivationType to a concrete type, and then create an element of that new type. Finally,
      in an instance of XML governed by this schema, the reference to key derivation in an instance of
      transformSec element would point not to instance of keyderivation element, but rather instance of the
      custom element. In other words, keyderivation would be SUBSTITUTED with a concrete key derivation element.
      In cases where default functionality is sufficient, the provided defaultKeyderivation element can be used for the
      substitution.


      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:element name = "keyderivation" type = "keyderivationType" substitutionGroup = "abstractKeyderivation">
    <xsd:annotation>
      <xsd:documentation>
        Default implementation of key derivation type.


      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name = "transformSecType">
    <xsd:annotation>
      <xsd:documentation>transformSecType: transformation information An element
      of transformsecType contains all of the information required to reverse the
      transformations applied to the original contents of the dataObject. It
      has two possible subsidiary elements: The algorithm element
      contains information about the algorithm used to encrypt the
      data. The key-derivation element contains the information that
      was used to derive the encryption key for this dataObject It has three
      attributes: 1. ID: an XML ID 2. transoformType: one of n predefined
      transformations types. Current valid types are compression,
      encryption, authentication. 3. order: If there are more than one
      transformation elements in an infoObjEntry this integer indicates
      the order in which the reversal transformations should be applied.


      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name = "algorithm" type = "xsd:string">
        <xsd:annotation>
          <xsd:documentation>algorithm element contains information
            about the algorithm used to encrypt the data.


          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element ref = "abstractKeyderivation" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" type = "xsd:ID"/>
    <xsd:attribute name = "order" type = "xsd:string"/>
    <xsd:attribute name = "transformType" use = "required">
      <xsd:simpleType>
        <xsd:restriction base = "xsd:string">
          <xsd:enumeration value = "COMPRESSION"/>
          <xsd:enumeration value = "AUTHENTICATION"/>
          <xsd:enumeration value = "ENCRYPTION"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name = "byteStreamType">
    <xsd:annotation>
      <xsd:documentation>byteStreamType: An element of  byteStreamType
      provides access to the current content of dataObjects for a XFDU
      document. The byteStreamType: has the following four attributes: ID (an XML ID);
      mimeType: the MIME type for the dataObject; size: the size of the dataObject
      in bytes; checksum: a checksum for dataObject; checksumType: type of checksum algorithms used to compute
checksum
      The data contained in these attributes is relevant to final state of data object after all possible transformations of the
original data.


      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name = "FLocat" type = "referenceType" minOccurs = "0" maxOccurs = "unbounded"/>
      <xsd:element name = "FContent" type = "fcontentType" minOccurs = "0"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
    <xsd:attribute name = "mimeType" type = "mimeType"/>
    <xsd:attribute name = "size" type = "xsd:long"/>
    <xsd:attribute name = "checksum" type = "xsd:string"/>
    <xsd:attribute name = "checksumType">
      <xsd:simpleType>
      <xsd:restriction base = "xsd:string">
        <xsd:enumeration value = "HAVAL"/>
        <xsd:enumeration value = "MD5"/>
        <xsd:enumeration value = "SHA-1"/>
        <xsd:enumeration value = "SHA-256"/>
        <xsd:enumeration value = "SHA-384"/>
        <xsd:enumeration value = "SHA-512"/>
        <xsd:enumeration value = "TIGER"/>
        <xsd:enumeration value = "WHIRLPOOL"/>
        <xsd:enumeration value = "CRC32"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name = "dataObjectType">
  <xsd:annotation>
    <xsd:documentation>dataObjectType : An element of dataObjectType
    contains current byteStream content and any required data to restore
    them to the form intended for the original designated community.
    It has two possible subsidiary elements: The byteStream element
    provides access to the current content dataObjects for an XFDU
    document. An element of dataObjectType must contain exactly 1 byteStream element
    that may contain an FLocat element, which provides a pointer to
    a content byteStream, and/or an FContent element, which wraps an
    encoded version of the dataObject. An element of dataObjectType may contain one or
    more transformation elements that contain all of the
    information required to reverse each transformation applied to
    the dataObject and return the original binary data object.
    The infoObjEntry has the following five attributes: 1. ID: an XML ID
    2, mimeType: the MIME type for the dataObject 3. size: the size of the dataObject
    in bytes 4. checksum: a checksum for dataObject 5. checksumType: type of checksum algorithms used to
    compute checksum 6. repID list of representation metadata IDREFs. The size, checksum, checksumtype and
    mime type are related to the original data before any transformations occured.


    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name = "byteStream" type = "byteStreamType"/>
    <xsd:sequence>
      <xsd:element name = "transformSec" type = "transformSecType" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:sequence>
  <xsd:attribute name = "ID" type = "xsd:ID"/>
  <xsd:attribute name = "repID" type = "xsd:IDREFS"/>
  <xsd:attribute name = "mimeType" type = "mimeType"/>
  <xsd:attribute name = "size" type = "xsd:long"/>
  <xsd:attribute name = "checksum" type = "xsd:string"/>
  <xsd:attribute name = "checksumType">
    <xsd:simpleType>
      <xsd:restriction base = "xsd:string">
        <xsd:enumeration value = "HAVAL"/>
        <xsd:enumeration value = "MD5"/>
        <xsd:enumeration value = "SHA-1"/>
        <xsd:enumeration value = "SHA-256"/>
        <xsd:enumeration value = "SHA-384"/>
        <xsd:enumeration value = "SHA-512"/>
        <xsd:enumeration value = "TIGER"/>
        <xsd:enumeration value = "WHIRLPOOL"/>
        <xsd:enumeration value = "CRC32"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name = "dataObjectSecType">
  <xsd:annotation>
    <xsd:documentation>dataObjectSecType : a container for one or more elements of dataObjectType


    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name = "dataObject" type = "dataObjectType" maxOccurs = "unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name = "contentUnitType">
  <xsd:annotation>
    <xsd:documentation>ContentUnit Complex Type The XFDU standard
    represents a data package structurally as a series of nested
    content units, that is, as a hierarchy (e.g., a data product,
    which is composed of datasets, which are composed of time
    series, which are composed of records). Every content node in
    the structural map hierarchy may be connected (via subsidiary
    XFDUptr or dataObjectPtr elements) to information objects which
    represent that unit as a portion of the whole package. The content
    units element has the following attributes:
    1.ID (an XML ID);
    2.order: an numeric string (e.g., 1.1, 1.2.1, 3,) representation
    of this unit's order among its siblings (e.g., its sequence);
    3.textInfo: a string label to describe this contentUnit to an end
    user viewing the document, as per a table of contents entry
    4.repID: a set of IDREFs to representation information sections
    within this XFDU document applicable to this contentUnit.
    5.dmdID: a set of IDREFS to descriptive information sections
    within this XFDU document applicable to this contentUnit.
    6.pdiID: a set of IDREFS to preservation description information
    sections within this XFDU document applicable to this
    contentUnit
    7.anyMdID: a set of IDREFS to any other metadata sections that do not fit
    rep,dmd or pdi metdata related to this  contentUnit
    88.unitType: a type of content unit (e.g., Application
    Data Unit, Data Description Unit, Software Installation Unit, etc.).
    contentUnitType is declared as a base type for concrete implementations of      contentUnit.


    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name = "XFDUptr" type = "referenceType" minOccurs = "0" maxOccurs = "unbounded">
      <xsd:annotation>
        <xsd:documentation>XFDUptr:XFDU Pointer. The XFDUptr element allows a
        content unit to be associated with a separate XFDU containing
```

```
          the content corresponding with that contentUnit, rather than
          pointing to one or more internal dataObjects. A typical instance of
          this would be the case of a thematic data product that collects
          data products from several instruments observe an event of
          interest. The content units for each instrument datasets might
          point to separate XFDUs, rather than having dataObjects and dataObject
          groups for every dataset encoded in one package. The XFDUptr
          element may have the following attributes: ID: an XML ID for
          this element locType: the type of locator contained in the
          xlink:href attribute; otherLocType: a string to indicate an
          alternative locType if the locType attribute itself has a value
          of "OTHER." xlink:href: see XLink standard
          (http://www.w3.org/TR/xlink) xlink:role: "" xlink:arcrole: ""
          xlink:title: "" xlink:show: "" xlink:actuate: "" NOTE: XFDUptr
          is an empty element. The location of the resource pointed to
          MUST be stored in the xlink:href element.


          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name = "dataObjectPtr" type = "dataObjectPtrType" minOccurs = "0" maxOccurs = "unbounded"/>
      <xsd:element ref = "abstractContentUnit" minOccurs = "0" maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" type = "xsd:ID"/>
    <xsd:attribute name = "order" type = "xsd:string"/>
    <xsd:attribute name = "unitType" type = "xsd:string"/>
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
    <xsd:attribute name = "repID" type = "xsd:IDREFS"/>
    <xsd:attribute name = "dmdID" type = "xsd:IDREFS"/>
    <xsd:attribute name = "pdiID" type = "xsd:IDREFS"/>
    <xsd:attribute name = "anyMdID" type = "xsd:IDREFS"/>
  </xsd:complexType>
  <xsd:element name = "abstractContentUnit" type = "contentUnitType" abstract = "true">
    <xsd:annotation>
      <xsd:documentation>abstractContentUnit is abstract implementation of
      contentUnitType. It cannot be instantiated in the instance
      document. Instead, concrete implementations would have to be
      used which are declared part of the contentUnit substitutionGroup


      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:element name = "contentUnit" type = "contentUnitType" substitutionGroup = "abstractContentUnit">
    <xsd:annotation>
      <xsd:documentation>contentUnit is basic concrete
        implementation of abstract conentUnit. Its instace can be used
        in the instance document in the place where contentUnit declared
        to be present.


      </xsd:documentation>
    </xsd:annotation>
  </xsd:element>
  <xsd:complexType name = "ipMapSecType">
```

```
    <xsd:annotation>
      <xsd:documentation>ipMapSecType Complex Type The Information Package Map
        Section (ipMapSec) outlines a hierarchical structure for the
        original object being encoded, using a series of nested
        contentUnit elements. An element of ipMapSecType has the following
        attributes: ID: an XML ID for the element; TYPE: the type of
        Information Product provided. Typical values will be"AIP" for a
        map which describes a complete AIP obeying all constrainsts and
        cardinalitiies in the OAIS reference model "SIP" for a map which
        describes a Submission Information Package textInfo: a string to
        describe the ipMapSec to users. packageType: a type for the object, e.g., book, journal, stereograph, etc.;
        Concrete implementation of contentUnit (defaultContentUnit, behavioralContentUnit, etc) have to be used in the
instance document.


      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref = "abstractContentUnit" maxOccurs = "unbounded"/>
    </xsd:sequence>
    <xsd:attribute name = "ID" type = "xsd:ID"/>
    <xsd:attribute name = "packageType" type = "xsd:string"/>
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
  </xsd:complexType>
  <xsd:complexType name = "interfaceDefType">
    <xsd:annotation>
      <xsd:documentation>interfaceDefType: interface definition object. The
        interface definition type contains a pointer an abstract
        definition of a set of related behaviors. These abstract
        behaviors can be associated with the content of a XFDU object.
        The interface definition element will be a pointer to another
        object (an interface definition object). An interface definition
        object could be another XFDU object, or some other entity (e.g.,
        a WSDL source). Ideally, an interface definition object should
        contain metadata that describes a set of behaviors or methods.
        It may also contain files that describe the intended usage of
        the behaviors, and possibly files that represent different
        expressions of the interface definition. An element of interfaceDefType
        is optional to allow for cases where an interface
        definition can be obtained from a behavior mechanism object (see
        the mechanism element of the behaviorSec).
        interfaceDef extends from referenceType and adds ability of specifying inputParameter
        that can be either just a string value or pointer to the content in this package


      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base = "referenceType">
        <xsd:sequence>
          <xsd:element name = "inputParameter" minOccurs = "0" maxOccurs = "unbounded">
            <xsd:complexType mixed = "true">
              <xsd:sequence>
                <xsd:element name = "dataObjectPtr" type = "dataObjectPtrType" minOccurs = "0"/>
              </xsd:sequence>
              <xsd:attribute name = "name" use = "required" type = "xsd:string"/>
```

```
                <xsd:attribute name = "value" type = "xsd:string"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name = "behaviorSecType">
      <xsd:annotation>
        <xsd:documentation>behaviorSecType: Complex Type for Behaviors. A
          behavior section can be used to associate executable behaviors
          with content in the XFDU object. A behavior section has an
          interface definition element that represents an abstract
          definition of the set of behaviors represented by a particular
          behavior section. A behavior section also has an behavior
          mechanism which is a module of executable code that implements
          and runs the behaviors defined abstractly by the interface
          definition. An behavior section may have the following
          attributes: 1. ID: an XML ID for the element 2. structID: IDREFS
          to structMap sections or divs within a structMap in the XFDU
          document. The content that the structID attribute points to is
          considered "input" to the behavior mechanism (executable)
          defined for the behaviorSec. 3. behaviorType: a behavior type
          identifier for a given set of related behaviors. 4. created:
          date this behavior section of the XFDU object was created. 5.
          textInfo: a description of the type of behaviors this section
          represents. 6. groupID: an identifier that establishes a
          correspondence between this behavior section and behavior
          sections. Typically, this will be used to facilitate versioning
          of behavior sections.  behavior  section may also include another behavior section for chaining of behaviors
          Concrete implementation of mechanism (wsdlMechanism,antMechanism,javaMechanism, etc) have to be used in
the instance document.


        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name = "interfaceDef" type = "interfaceDefType" minOccurs = "0"/>
        <xsd:element ref = "abstractMechanism"/>
        <xsd:element name = "behaviorSec" type = "behaviorSecType" minOccurs = "0" maxOccurs = "unbounded"/>
      </xsd:sequence>
      <xsd:attribute name = "ID" use = "required" type = "xsd:ID"/>
      <xsd:attribute name = "structID" use = "required" type = "xsd:IDREFS"/>
      <xsd:attribute name = "behaviorType" type = "xsd:string"/>
      <xsd:attribute name = "created" type = "xsd:dateTime"/>
      <xsd:attribute name = "textInfo" type = "xsd:string"/>
      <xsd:attribute name = "groupID" type = "xsd:string"/>
    </xsd:complexType>
    <xsd:element name = "abstractMechanism" type = "mechanismType" abstract = "true">
      <xsd:annotation>
        <xsd:documentation>abstractMechanism is abstract implementation of
          mechanismType. It cannot be instanciated in the instance
          document. Instead, concrete implementations would have to be
          used which are declared part of mechanism substitutionGroup
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:complexType name = "mechanismType">
      <xsd:annotation>
        <xsd:documentation>mechanismType: executable mechanism. An element of  mechanismType
          contains a pointer to an executable code module that
          implements a set of behaviors defined by an interface
          definition. The mechanism element will be a pointer to another
          object (a mechanism object). A mechanism object could be another
          XFDU object, or some other entity (e.g., a WSDL source). A
          mechanism object should contain executable code, pointers to
          executable code, or specifications for binding to network
          services (e.g., web services).
          mechanismType is declared as base type for concrete implementations of mechanism


        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexContent>
        <xsd:extension base = "referenceType"/>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name = "wsMechanism" type = "mechanismType" substitutionGroup = "abstractMechanism">
      <xsd:annotation>
        <xsd:documentation>wsMechanism is concrete implementation of
          abstract mechanism which implements mechanism based on Web
          service. Its instace can be used in the instance document in the
          place where mechanism declared to be present.


        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>
    <xsd:element name = "javaMechanism" substitutionGroup = "abstractMechanism">
      <xsd:annotation>
        <xsd:documentation>javaMechanism is concrete implementation of
          abstract mechanism which implements Java-based mechanism.
          xlink:href element points to fully qualified name of the Java
          class that implements the mechanism. FLocate element specifies
          location of the jar file where that class is packaged. File can
          be local to this XFDU package, or located on the remote server,
          or somewhere on the local file system.An instace of
          javaMechanism can be used in the instance document in the place
          where mechanism declared to be present.


        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base = "mechanismType">
            <xsd:sequence>
              <xsd:element name = "FLocat" type = "referenceType"/>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name = "antMechanism" substitutionGroup = "abstractMechanism">
      <xsd:annotation>
        <xsd:documentation>antMechanism is concrete implementation of
          abstract mechanism which implements ANT-based mechanism.
          xlink:href element points to a location of ANT script to be
          executed. Also, xmlData element can contain ANT specific XML. An
          instace of antMechanism can be used in the instance document in
          the place where mechanism declared to be present.


        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base = "mechanismType">
            <xsd:sequence>
              <xsd:element name = "xmlData" type = "xmlDataType" minOccurs = "0"/>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:complexType name = "metadataSecType">
      <xsd:sequence>
        <xsd:element name = "metadataObject" type = "mdSecType" minOccurs = "0" maxOccurs = "unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name = "XFDUType">
      <xsd:annotation>
        <xsd:documentation>
        XFDUType Complex Type.
        A XFDU document consists of five possible subsidiary sections:
        packHdr (XFDU document header), dmdMD (descriptive metadata
        section), repMD (representation  metadata section), pdiMD (preservation information section),
        infoObjEntrSec (data object section),ipMapSec (content unit section), behaviorSec (behavior section).
        It also has possible attributes:
        1. ID (an XML ID);
        2. objID: a primary identifier assigned to the original source   document;
        3. textInfo: a title/text string identifying the document for   users;
        5. version: version to which this XFDU document conforms


        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name = "packHdr" type = "packHdrType" minOccurs = "0"/>
        <xsd:element name = "metadataSec" type = "metadataSecType" minOccurs = "0"/>
        <xsd:element name = "dataObjectSec" type = "dataObjectSecType" minOccurs = "0"/>
        <xsd:element name = "IpMapSec" type = "ipMapSecType"/>
        <xsd:element name = "behaviorSec" type = "behaviorSecType" minOccurs = "0" maxOccurs = "unbounded"/>
      </xsd:sequence>
      <xsd:attribute name = "ID" type = "xsd:ID"/>
      <xsd:attribute name = "objID" type = "xsd:string"/>
```

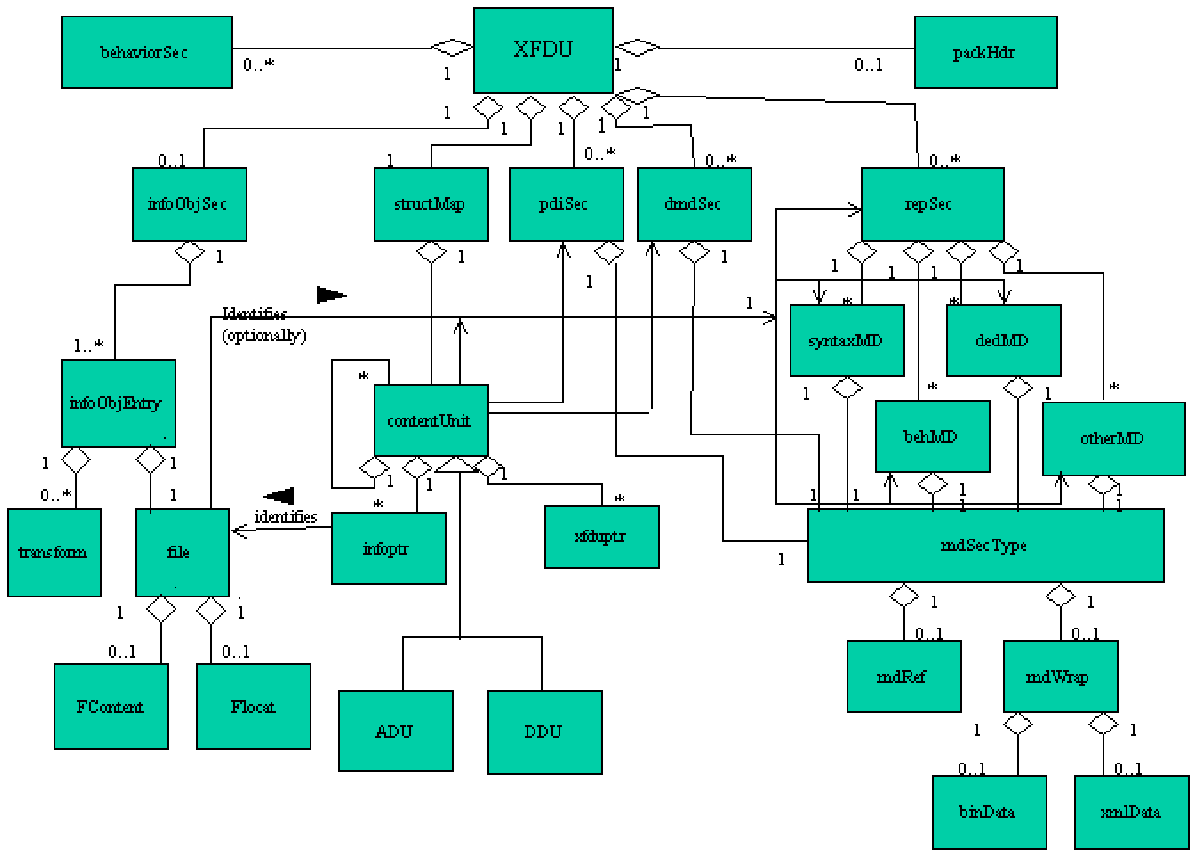
```
    <xsd:attribute name = "textInfo" type = "xsd:string"/>
    <xsd:attribute name = "version" type = "xsd:string"/>
  </xsd:complexType>
  <xsd:element name = "XFDU" type = "XFDUType"/>
</xsd:schema>
```

Annex 1 Example XFDU

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XFDU>
  <packHdr>
    <envirMD ID="envirMD1">All platforms</envirMD>
    <transformMD algorithmName="blowfish" mustUnderstand="true"
description="encryption"/>
  </packHdr>
  <dmdSec ID="ECSDMD">
    <mdWrap mdType="OTHER">
      <xmlData>
        <dmdMD>some ECS defined XML</dmdMD>
      </xmlData>
    </mdWrap>
  </dmdSec>
  <repSec ID="mathMLAlgRepSec">
    <otherMD ID="mathMLMD">
      <mdWrap mdType="OTHER" textInfo="mathML encoding of the algorithm">
        <xmlData>
          <math>
            <mrow>
              <mo>det</mo>
              <mo symmetric="false" rspace="0" lspace="0">|</mo>
              <mfrac linethickness="0">
                <mi>a</mi>
                <mi>c</mi>     </mfrac>
              <mfrac linethickness="0">
                <mi>b</mi>
                <mi>d</mi>     </mfrac>
              <mo symmetric="false" rspace="0" lspace="0">|</mo>
              <mo>=</mo>
              <mi>a</mi>
              <mi>d</mi>
              <mo>-</mo>
              <mi>b</mi>
              <mi>c</mi>
              <mo>,</mo> </mrow>
          </math>
        </xmlData>
      </mdWrap>
    </otherMD>
  </repSec>
  <repSec ID="ATD">
    <otherMD ID="atdMD">
      <infoObjEntryPtr infoObjEntryID="ATD MD"/>
    </otherMD>
  </repSec>
  <pdiSec ID="pdiSec">
    <provenanceSec ID="provenance">
      <mdRef mdType="OTHER" mimeType="text/xml" textInfo="processing  history  XML file" locType="URL"
ns1:href="file:packagesamples/scenario1/pdi.xml"
xmlns:ns1="http://www.w3.org/TR/xlink"/>
    </provenanceSec>
  </pdiSec>
  <infoObjEntrSec>
    <infoObjEntry repID="mathMLMD" ID="mpeg21">
      <dataObject  mimeType="video/mpeg" checksum="b3eb4b34" ID="mpeg21AnimData" checksumType="CRC32" size="414131">
        <FLocat locType="URL"
ns2:href="file:packagesamples/scenario1/mpeg21.mpg" xmlns:ns2="http://www.w3.org/TR/xlink"/>
      </dataObject>
    </infoObjEntry>
    <infoObjEntry size="0" checksumType="CRC32" checksum="6d0e30ea"
mimeType="application/pdf" ID="ATD MD">
      <dataObject mimeType="application/octetstream" checksum="ad78ad5d"
ID="atdMD" checksumType="CRC32" size="110874">
        <FLocat locType="URL" ns3:href="file:packagesamples/scenario1/atd.pdf"
```

```
xmlns:ns3="http://www.w3.org/TR/xlink"/>
          </dataObject>
          <transformsec type="ENCRYPTION">
             <algorithm>blowfish</algorithm>
          </transformsec>
       </infoObjEntry>
       <infoObjEntry repID="atdMD" ID="hdfFile0">
          <dataObject mimeType="application/x-hdf" checksum="acab6535"
ID="hdfFile0" checksumType="CRC32" size="10455471">
             <FLocat locType="URL"
ns4:href="file:packagesamples/scenario1/mod1.hdf"
xmlns:ns4="http://www.w3.org/TR/xlink"/>
          </dataObject>
       </infoObjEntry>
       <infoObjEntry repID="atdMD" ID="hdfFile1">
          <dataObject mimeType="application/x-hdf" checksum="acab6535"
ID="hdfFile1" checksumType="CRC32" size="10455471">
             <FLocat locType="URL"
ns5:href="file:packagesamples/scenario1/mod2.hdf"
xmlns:ns5="http://www.w3.org/TR/xlink"/>
          </dataObject>
       </infoObjEntry>
       <infoObjEntry repID="atdMD" ID="hdfFile2">
          <dataObject mimeType="application/x-hdf" checksum="acab6535"
ID="hdfFile2" checksumType="CRC32" size="10455471">
             <FLocat locType="URL"
ns6:href="file:packagesamples/scenario1/mod3.hdf"
xmlns:ns6="http://www.w3.org/TR/xlink"/>
          </dataObject>
       </infoObjEntry>
       <infoObjEntry mimeType="application/octetstream" ID="orbitalData">
          <dataObject checksum="b3eb4b34" ID="orbitData" checksumType="CRC32">
             <FLocat locType="URL"
ns7:href="http://coin.gsfc.nasa.gov:8080/ims-bin/3.0.1/nph-ims.cgi?msubmit=yes&a
mp;lastmode=SRCHFORM" xmlns:ns7="http://www.w3.org/TR/xlink"/>
             <FContent>
               <binData>UEsDBBQACAAIAKqMBC8AAAAAAAAAAAAAAAAPAAAAeGZkdS8uY2xhc3NwYXRotZXfS8MwEMff/StK35OugqCwH4hO0I...
             </binData>
             </FContent>
          </dataObject>
       </infoObjEntry>
    </infoObjEntrSec>
    <ipMapSec>
       <defaultContentUnit repID="atdMD" pdiID="provenance" dmdID="ECSDMD">
          <infoObjEntryPtr infoObjEntryID="mpeg21"/>
          <defaultContentUnit order="1" textInfo="Root content unit for HDF data">
             <defaultContentUnit order="1.1" pdiID="provenance" textInfo="content unit for hdfFile0" dmdID="ECSDMD">
                <infoObjEntryPtr infoObjEntryID="hdfFile0"/>
             </defaultContentUnit>
             <defaultContentUnit order="1.2" pdiID="provenance" textInfo="content unit for hdfFile1" dmdID="ECSDMD">
                <infoObjEntryPtr infoObjEntryID="hdfFile1"/>
             </defaultContentUnit>
             <defaultContentUnit order="1.3" pdiID="provenance" textInfo="content unit for hdfFile2" dmdID="ECSDMD">
                <infoObjEntryPtr infoObjEntryID="hdfFile2"/>
             </defaultContentUnit>
          </defaultContentUnit>
          <defaultContentUnit textInfo="content unit for orbit data">
             <infoObjEntryPtr infoObjEntryID="orbitalData"/>
          </defaultContentUnit>
       </defaultContentUnit>
       <defaultContentUnit textInfo="content unit ATD metadata">
          <infoObjEntryPtr infoObjEntryID="ATD MD"/>
       </defaultContentUnit>
    </ipMapSec>
</XFDU>
```

Annex 2 UML for XFDU – Needs to be updated



Annex 3 Relationship to Other Efforts

A primary goal of this effort was to reuse concepts and constructs from other efforts (commercial, academic, and standards) to define methods to package multiple types (e.g., binary, text, audio, video, etc) of data and metadata for both archival and transmission. There were several efforts identified. These efforts varied significantly in many aspects including maturity, existing implementations and intended scope of designated community. None of the products adequately met the requirements defined in the Annex 2 to allow adoption of the existing specifications. The following is a brief summary of the efforts that had a major influence on this recommendation and an overview of the areas of this recommendation that were adapted from each effort. More detailed discussions of design tradeoffs may be found or referred to in the appropriate section of this recommendation.

XML PACKAGING TF –W3C

In August 1999 W3C released a report from a task force on XML packaging that had been commissioned but discontinued due to member priorities. This report provides an excellent overview of the area and divides the problem into two basic efforts:

1. the design of a **manifest,** an XML structure to describe the associations among resources and metadata and the locations of those resources and
2. the underlying packaging mechanism that groups the various resources and metadata into a single unit

The W3C Packaging Committee makes these comments on use of XML as the packaging technique:
XML is an obvious mechanism to consider when considering a packaging mechanism. XML is simple. It should not take to long to design a mechanism that uses XML. It allows us to use a standard tool to do much of the work. Many have suggested XML as a packaging mechanism. Problems with XML:

- Would have to extend XML to allow true binary data.
- XML has far less field experience than either MIME or ZIP, especially in the area of embedding binary data.
- XML is ill-suited for random access. White space handling, entity expansion, and the intrinsic expectation that XML is serially scanned from beginning to end work against random access. Canonicalization can help some of these problems, but this may be too great a burden for implementors.
- XML is not the most concise representation.

The W3C Packaging Committee makes these comments on use of File/Directory Compression Techniques:
ZIP is a highly used format for packaging information for transmission on the web. Other packaging formats are based on ZIP, such as JAR, Java Archive, and CAB, Cabinet files. The following paragraphs contains information on ZIP, some of which is the ZIP file format specification, and another part being the source code for zip and unzip:
ftp://ftp.uu.net/pub/archiving/zip

It is understood that as long as InfoZIP's copyright is left in place, the working group can do more or less as it pleases. It is also believed that nobody else has credible intellectual property claims to either the code or the algorithms that are used by ZIP. A quote from ftp://ftp.uu.net/pub/archiving/zip/doc/COPYING says: "In other words, use it with our blessings, but it's still our [InfoZIP's] code. Thank you!"

At least one publisher has implemented the Zip packaging technology for electronic distribution of book files. They have used a proprietary encryption scheme for component files, while leaving the Zip structure in the clear to get around the direct access to component files problem. Problems with ZIP:

- File name character encoding limitations (not Unicode)
- Not a true hierarchy, faked with slashes in file names.
- Mandatory index (comes last, though).
- Inefficient for on-disk editing.

METS --DLF

METS is the result of a Digital Library Federation initiative, and attempts to provide an XML document format for encoding metadata necessary for both management of digital library objects within a repository and exchange of such objects between repositories (or between repositories and their users). Depending on its use, a METS document could be used in the role of a Submission Information Package (SIP), Archival Information Package (AIP), or Dissemination Information Package (DIP) within the Open Archival Information System (OAIS) Reference Model. The METS schema provides a flexible mechanism for encoding descriptive, administrative, and structural metadata for a digital library object, and for expressing the complex links between these various forms of metadata.

Given the close working relationship between the digital library community and CCSDS Panel 2 on the OAIS RM and the fact that the METS Information model was based on OAIS concepts it was suggested that the METS schema be inherited as the basis for this effort.

METS is a very flexible structure that has been developed by the Digital Library with some attention to the OAIS RM. The metadata and file association methods are very flexible and allow nearly direct translation of SFDU labels and ADIDs .for identifying, locating and making a Repository Service Request for a detached data description in EAST or XDF. However, METS is more of a conceptual model and the Representation Data mapping is questionable. The proposed XFDU Data Model should maps directly to the OAIS Information Model Classes.
.

XPACK – ESA/ANITE

The XPack XML Schema was developed by Anite for ESA as a portion of a study on the use of XML as a packaging technology in the ESA Data Distribution System. This prototype included the design of an XML Schema describing a vocabulary that can be used to package arbitrary data (plain text, XML, binary, etc.) and metadata and establish relationships between the data and the metadata through the use of SFDU classes. This schema focused on using a single XML document as the package but did allow in line content to be replaced by Xlink hrefs. The vocabulary also supports services that are commonly required in association to data packaging, including compression, encryption, authentication, fragmentation, and validation.

OPEN OFFICE XML FILE FORMAT– SUN

The newest major release of OpenOffice/StarOffice from SUN et al uses XML as its native format and a packaging technique based on ZIP with a well-documented manifest file. This packaging methodology is based on very well proven XML standards but includes support of a complex encryption technique. Though this application does not require the complex metadata linkage required by space related data or long-term preservation, it is significant in that they are a large commercial project using an XML/ZIP packaging techniques with encryption and binary files on many platforms

XPACKAGE -OPEN EBOOK

The OeBF Publication Structure Working Group is defining an XML Package format (XPackage) to be the basis in part of the OeB Publication Structure 2.0. for defining packages, or collections of resources and their associations. It specifies a framework for describing the resources which are included in such packages, the properties of those resources, their method of inclusion, and their relationships between each another. XPackage use cases include specifying the stylesheets used by an HTML document, declaring the images shared by multiple documents, indicating the author and other metadata of a document, describing how namespaces are used by XML resources, illustrating fallback sequences for varying levels of multimedia support, and providing a manifest for bundling resources into a single archive file. The XPackage framework is based upon XML, RDF, and XLink, and provides two RDF ontologies: one for general packaging descriptions and another for describing XML-based resources.

XPackage represents a leading edge use of W3C standards to find a conceptual solution to the description of the relationships among element of a package.

GLOBUS PACKAGING TOOLKIT

The Grid Packaging Tool (GPT) uses XML to describe all the information that is needed for an installation. These XML files hold all of the information that the GPT will need in order to create a Grid instance on a particular system. The intent of this XML is to construct a framework and a set of Grid packages that can be used to create tailored Grid to meet individual needs. This will allow for the distribution and release of individual Grid components, rather then a single monolithic release. This concept will allow organizations to construct both source and binary distributions of packages they are interested in rather then being forced to build and configure components that are of no interest.

Given the importance of software as representation data , the cooperation or the GRID , the Space Science community, the XML based approach used by the GPT and the experience being gained in the releases of the Globus Service Toolkit, any software distribution unit in the XFDU should be exactly GPT if possible. The GPT DTD and element descriptions appear as chapter 15.

Annex 4

12.1 PHYSICAL PACKAGING TECHNIQUES

The XFDU package is a container that contains an XFDU document and a set of data objects. There are three common types of container object that could be supported:

- Archive formats (such as zip, jar or tar), which are already widely deployed, may be used as container.
- Message formats (such as Soap with Attachments**[REF3]** and Direct Internet Message Encapsulation (DIME)**[REF4]**) which are a major focus of XML protocol and eBusiness efforts.
- XML document format The XFDU document can be considered as both the manifest and a container for ASCII/XML files or binary data encoded using XML Schema approved techniques.

This section briefly discusses these alternatives and makes recommendations for this effort

12.1.1 FILE/DIRECTORY COMPRESSION TECHNIQUES

A packaging technique that has been successfully used by commercial efforts is the consolidation of the entire directory structure, including all files, into a single file. The current implementations use ZIP, but others such as CAB, JAR, and TAR can also be used. A well-known "**manifest**"file with a specific name in a specific directory provides the locations and associations of the package resources.

12.1.2 SINGLE XML DOCUMENT

This technique is the simplest of the techniques. It simply uses an encoding scheme to encode any binary data into valid XML characters and store all content inline. This solution does not seem to be optimal for applications or domains with large binary content but does enable the use of standard XML tools.

12.1.3 WEB SERVICE MESSAGE TECHNIQUES

The emergence of XML Web Services and eBusiness has created a requirement for a single XML message to encapsulate multiple business object types including binary objects such as spreadsheets and documents created on word processors.In the HTML and email domains this problem is solved using Multipurpose Internet Mail Extensions (MIME) One of the primary uses of MIME types is to determine the "helper application" should process the contents of the files.There are MIME types for most major formats, formats, both binary and textuall There are also MIME types to define the format of the messages that encapsulate the attachments.For example modern email software use standard MIME types to Attach multiple files to a single email.The Soap with Attachments efforts are based on adapting the MIME types to the SOAP XML envirnment. Specifically, he specification combines specific usage of the Multipart/Related MIME media type RFC 2387 and the URI schemes discussed in RFC 2111 and RFC2557 for referencing MIME parts.

12.2 USE OAIS INFORMATION MODEL

METS (Metadata Encoding and Transmission Standard) is a very flexible structure that has been developed by the Digital Library Foundation with some attention to the OAIS RM. METS incorporates many useful mechanisms that have served as starting points in the development of this schema; however, METS is more of a conceptual model I don't know what you mean by 'conceptual model' here. I don't usually think of METS as a conceptual model – only as a schema, which is an implementation of sorts, and I'm used to opposing 'implementation' to 'conceptual model' – so I assume you mean it in some other way, but I don't know what that way is. and the metadata classification especially Representation Data mapping is questionable. The proposed XFDU Data Model should map directly to the OAIS Information Model Classes.

12.2.1 MANDATORY MANIFEST/TABLE OF CONTENTS

XPACK emphasized the single document, pure XML approach, with binary data encoded as base 64 within the document. Using this approach, a table of contents was optional. However, W3C Packaging Task Force Recommendation, the OpenOffice implementation, and the METS schema all separate the manifest from the data objects. The decision for Release 1 of the XFDU is to have a mandatory XML encoded map of the information package and metadata to enable a common schema for all packaging forms.

12.2.2 FLEXIBLE LINKAGE OF MANIFEST TO DATA AND METADATA

The XFDU Manifest must contain flexible methods to reference/include the data objects and metadata about those objects. The METS standard offers very flexible metadata/data linkage and referencing methods. We have adopted the METS mechanisms that enable:

- Data Objects that are contained in the manifest are to be encoded in base64 or XML
- Data Objects that are included by reference from the manifest are to exist as files in the XFDU package or as files with known URIs either in a repository or in a location accessable via URL
- Metadata objects that are contained in the manifest are to be encoded in base64 or XML
- Metadata objects that are included by reference from the manifest are to exist as files in the XFDU package or as files with known URIs either in a repository or in a location accessable via URL
- Information Objects can reference applicable Metadata objects by ID where the name of the referencing attribute is used to classify the Metadata and the schema enables identification of the source of the metadata

In addition to the METS capabilities detailed above ,the XFDU structure allows metadata objects to be treated as data objects. This enables direct mapping to the OAIS representation net where each metadata object is an information object containing both data object and representation information.

12.2.3 RELATIONSHIP DESCRIPTION

A major component of information packaging is the description of the relationships among the various metadata and data objects in both human readable and machine interpretable forms. The "Semantic Web" efforts have led to several XML notations for expressing relationships and rules among resources. The two main efforts reviewed in this area were Resource Description Format (RDF) and Web Ontology Language (OWL). RDF has been a World Wide Web Consortium (W3C) Recommendation for several years but software support and automated applications have lagged the recommendation considerably. OWL has recently been promoted to a W3C Candidate Recommendation. Based on experience with other complex W3C Recommendations, this means that a final recommendation is expected in 1-2 years. However, OWL is derived from several existing ontology languages, so there is significant legacy software that can quickly be modified to the OWL syntax.

Due to the lack of mature XML software products in this area, the initial version of this recommendation will not use a formal XML grammer to express the range of potential relationships. Instead, the XFDU schema uses the OAIS information model to classify information objects and content unit types to define the overall structure and semantics of contained objects. Alsothe complete Xlink schema is included in the XFDU referencing mechanisms to allow the use of complex Xlink attribues to model resource relationships.

It is anticipated that Future versions of this recommendation will use an XML based language such as OWL to improve description and enable software services for generic relationships.

12.2.4 EXTENSIBILITY MECHANISMS

In order to allow an orderly evolution of the XFDU schema and to allow the development of specialized versions of complex elements to enable additional functionality or alternative implementations during prototyping
There are a number of XML Schema mechanisms that can be used to achieve the goal:

5. Use of an abstract element and element substitution using substitutionGroup
6. Use of abstract type and type substitution using xsi:type
7. Use of a <choice> element
8. Use of a dangling type

The use of dangling types is theoretically the most flexible solution, but is not supported by most XML parsers. The use of choice is the least attractive solution because it does not support the use of complex elements and creates rigid schemas.

Type substitution is slightly superior to element substitution from a schema design viewpoint; however, there are two factors that inform the decision to use element substitution for this version. The first is the visibility of xsi:type in every XML instance. This has proven unpopular with users of other standard schemas. The second factor is that some early experimentation with inputting XML Schema to JAVA tools revealed that element substitution was partially supported while type substitution was not supported.

Annex 4 Use Cases

The XML workshop held at GSFC in the summer of 2001 defined two use cases to illustrate the anticipated usage and functionality of the next generation of CCSDS Packaging mechanisms.

Use Cases Defined At CCSDS XML 2000 Workshop

Simple ASCII DATA Transfer

- has simple ASCII product (XML file) – orbit file for 1 month
- data description is available as XML schema
- Data Dictionary available
- Wants to ship to archive (negotiations completed)
- Uses CCSDS XML Package
- Send single document

Data Provider

1. Gets CCSDS Package schema in order to make instance
2. Uses some XML editor to fill out required fields in XML file including
- Copies in Data Dictionary into XML document
- *[Could point to external dictionary using XPOINTER]*
- Copies Orbit file data (already XML) into XML document
- Copies data description schema into XML document
- *[Could just point to it if held somewhere external]*
- Creates metadata based on metadata schema agreed with archive
- Schema is registered and held somewhere
- E.g. Satellite name in some standard/agreed way
- Start and Stop times
- Orbit number(s)
- Etc etc
3. Send as single XML file

Archive:
- receives XML file from producer
- verifies safe receipt e.g. checksum, signing (??)
- may also authenticate the source of the data if needed
- Parse and Validate XML against schema.
- checks whether or not the file is self-contained or refers to external files
- gather any external files needed
- Repackages all the information together ready to archive
- Could use XSLT to repackage and also to separate descriptive information to go into catalogues etc

SENDING LARGE BINARY FILES (e.g. telemetry)

- large binary file (200MB)
- data description is available as EAST description
- Data Dictionary available
- Wants to ship to archive (negotiations completed)
- Uses CCSDS XML Package

<u>PRODUCER:</u>

1 Gets CCSDS Package schema in order to make instance
2 Uses some XML editor to fill out required fields in XML file including
- Copies in Data Dictionary into XML document
- *[Could point to external dictionary using XPOINTER*
- Add pointer (relative URI - local file name) for data into XML document
- Add pointer to EAST description which is held in Metadata registry
- Creates metadata based on metadata schema agreed with archive
- Schema is registered and held somewhere
- E.g. Satellite name in some standard/agreed way
- Start and Stop times
- Orbit number(s)
- Unique ID
- List of files (Manifest?)
- Etc etc

3 Zips up all the files (XML and Binary files)
4 Sends to archive

<u>ARCHIVE:</u>

1. receives (ZIP) file from producer
2. Recognise that it is a ZIP file
3. verifies safe receipt e.g checksum, signing (??)
4. may also authenticate the source of the data if needed
5. UNZIP
6. Find STANDARD NAMED file as root e.g "**QWERTY.XML**"
7. Parse the **QWERTY.XML**
- Parse and Validate XML against schema.
- checks whether or not the file is self-contained or refers to external files
- gather any external files/packages needed (e.g. EAST description)
- Repackages all the information together ready to archive
- Could use XSLT to repackage and also to separate descriptive information to go into catalogues etc

8. Archive wants to extract number from binary file
- Needs to recognise that EAST interpreter is needed
- Starts EAST interpreter
- Gives data file
- Gives EAST description
- Matches name of item wanted to Data Dictionary e.g. ALIAS
- Matches Data Dictionary name to EAST description
- Use EAST interpreter to extract number
- DONE

8. DONE

Annex 5 Requirements

User Requirements

Major capabilities

1. The Package is a container which can contain a single object or a collection of objects which may be organised as a Hierarchical structure of objects (HSO)
2. A package may contain other packages
    a) outer package (the file interchanged) is identified as a package by external means
    b) may be problem with uniqueness of outer "manifest"
    c) The outer package MAY be e.g. ZIP or pure XML or encrypted etc
3. Each object and grouping of objects should be accompanied by appropriate metadata including:
    a) EITHER the description (Representation Information) OR an identification of the description of the object
    b) Identification of the type of the objects
4. Objects may be character or binary or both
5. *An object is ……..(a document - NO)*
6. An object or package may be contained in a single file or multiple files
    a) Need to distinguish between packages which are logically self-contained and those which are physically self-contained i.e. single files
    b) Allow some of the objects to be separate files on the same medium (e.g. spans files) or may be distributed at different sites on a network
7. Each object may be assigned an identifier which is unique within the package
8. Need the ability to have universally (*within some explicitly agreed domain*??) unique identifier
    a) Version of the Identifier (TBC)

Additional capabilities

9. A mechanism to contain relationship information and identify the objects involved
    a) these relationships may be among objects within the same package, and between/among those *outside* and inside of a package.
    b) also a 'table of contents' of the package, which requires the ability to point to individual objects at various levels of nesting within containers - MANIFEST mechanism
    c) allow lists and alternate views
10. A mechanism to identify that, in addition to the description of the object or package, there may a number of uniquely identified decodings that should be applied in a particular sequence to reverse the encodings that have been applied. Example: a object has been encoded and then compressed, or that a set of objects has been tarred and then compressed.
    a) Allow encodings to be combined without having to register each combination. This will allow encodings to be readily changed during processing of the package, without having to re-do registrations of descriptions.
    b) Some mechanism needed to specify process/application to process the description/encoding information
    c) Provide a mechanism to hold a description of the encodings
    d) Some mechanism needed to be able to specify encoding of the package as a whole
        ▪ There must be the ability to identify the start of the package or object

11. *The package should be easily usable within applications* – i.e. at least one well-defined API/Interface should be defined.
12. Need mechanism(s) to allow one to verify integrity & authenticity of the whole or parts of the package where required
13. *Need to be able to do "lite" packaging i.e. not too many mandatory items.*
14. Be able to support use of multiple (i.e. some) registries
15. *May wish to have the ability to begin processing a single package BEFORE it is all received.*

Further Issues

- *Relationships*
  - *What types of pre-defined relationships should we support?*
- *High level packaging*
- *Encoding information*
- *Pointer mechanisms*
  - *implied*
  - *Explicit identification of pointer mechanisms*
- *OASIS protocol interface – implications on packaging*
- *Packaging of binary objects*
  - *Internal*
  - *External*
  - *Performance*
  - *Mixing binary with text in same object*
- *Description "technology" specification*

Annex 2 Functional Requirements

SR1 Description: The XPack document type will support the packaging of files.
Source: UR1
Importance: High

SR2 Description: The XPack document type will provide a mechanism for embedding files directly into an XPack document.
Source: UR1
Importance: High

SR3 Description: The XPack document type will provide a mechanism for referring to external files which make up part of a package.
Source: UR1
Importance: High

SR4 Description: The XPack document type will support the packaging of raw data.
Source: UR1
Importance: High

SR5 Description: The XPack document type will directly support or allow external document model extensions which support the notion of content
datatype specification.
Source: UR1
Importance: High

SR6 Description: The XPack document type will provide elements which can be repeated to allow multiple objects to be contained within the
document.
Source: UR4
Importance: High

SR7 Description: The XPack document type will provide elements or attributes which indicate that the package represented by the document is distributed
across one or more files.
Source: UR5
Importance: High

SR8 Description: The XPack document type will allow its root element to be nested.
Source: UR6
Importance: High

SR9 Description: The XPack document type will provide elements or attributes which allow metadata about the package to be defined.
Source: UR7
Importance: High

SR10 Description: The XPack document type will provide elements or attributes which will allow the definition of references to metadata registries which contain metadata describing the objects stored in a package.
Source: UR8
Importance: High

SR11 Description: The XPack document type will provide elements or attributes which enable lists of files to be defined and referenced.
Source: UR9
Importance: High

SR12 Description: The XPack document type will provide elements which can be nested to define a hierarchical structure of packaged objects.
Source: UR10
Importance: High

SR13 Description: It will be possible to associate the XPack elements and attributes which support metadata definition and referencing to each packaged object in an XPack document.
Source: UR11
Importance: High

SR14 Description: The XPack document type will provide elements or attributes which enable the definition of object labels describing the class, type, and version of an object.
Source: UR12
Importance: Medium

SR15 Description: The XPack document type will provide elements or attributes which allow an object label to be associated with an object in an XPack document.
Source: UR12
Importance: Medium

SR16 Description: The XPack document type will provide an element for packaging character data.
Source: UR13
Importance: High

SR17 Description: The XPack document type will provide an element for packaging binary data in a character encoded format.
Source: UR13
Importance: High

SR18 Description: The XPack document type will provide an attribute which allows

each object contained in a package to have a globally unique
identifier assigned to it.
Source: UR14
Importance: Medium

SR19 Description: The XPack document type will recommend an algorithm that should
be used for generating globally unique identifiers.
Source: UR14
Importance: Medium

SR20 Description: The XPack document type will provide elements or attributes which
may be used to specify relationship information about the relationships between objects contained in a package.
Source: UR20
Importance: High