Implementing OAIS Reference Model Concepts: Information Packages and Producer-Archive Interface Standards

Lou Reich CSC/NASA,

Chairman of the CCSDS information Packaging and Registries Working Group

Don Sawyer NASA/CCSDS

Co-Chairman of the CCSDS Data Archive Ingest (DAI Working Group)
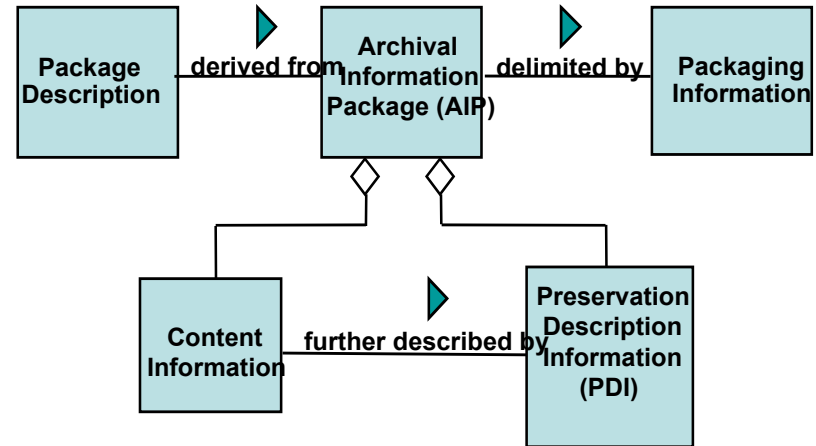
10 November 2006

DLF Forum

Boston Mass

In the Beginning: OAIS Reference Model
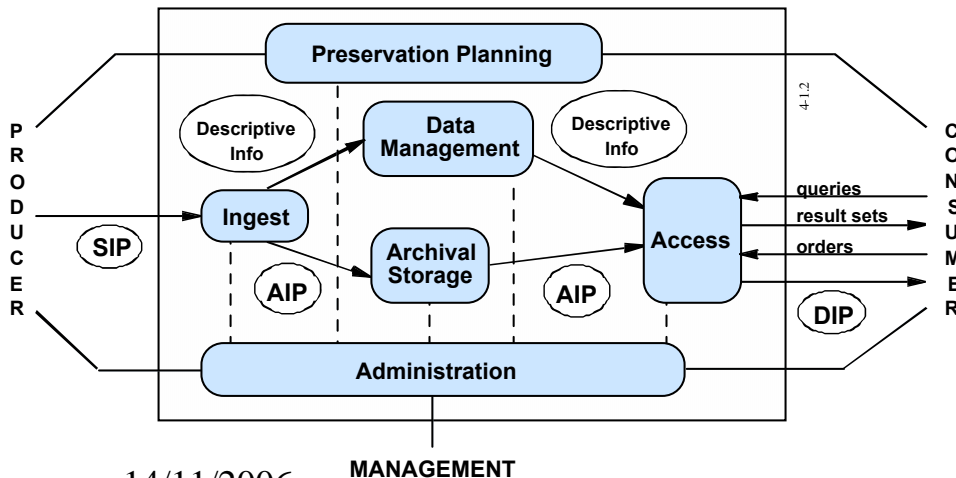
OAIS Mandatory Responsibilities:

1. **Negotiating and accepting information**
2. **Obtaining sufficient control of the information to ensure long-term preservation**
3. **Determining the "designated community"**
4. **Ensuring that information is independently understandable**
5. **Following documented policies and procedures**
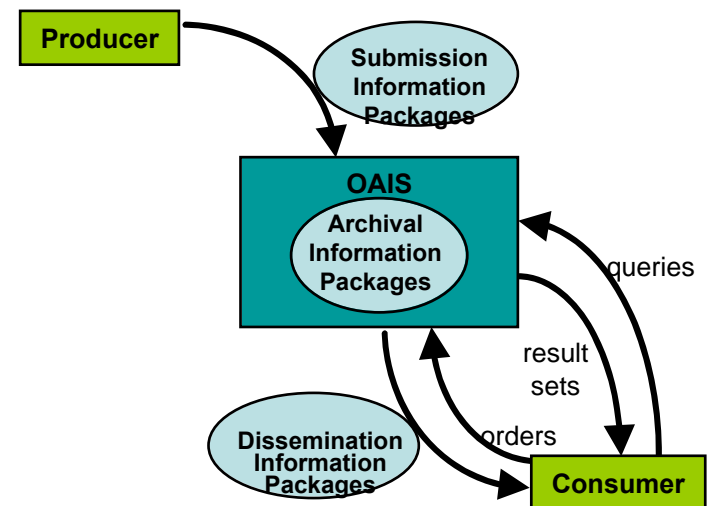6. **Making the preserved information available**

OAIS Information Model

```
Package          derived from    Archival      delimited by    Packaging
Description      ───────────►    Information    ───────────►    Information
                                 Package (AIP)

                          ◇            ◇

Content          further described by      Preservation
Information      ─────────────────────►    Description
                                            Information
                                            (PDI)
```

OAIS RM

OAIS Functional Model

P R O D U C E R

SIP

Preservation Planning

Descriptive Info — Data Management — Descriptive Info

Ingest

Archival Storage

Access

AIP   AIP

Administration

queries
result sets
orders

C O N S U M E R

DIP

4-1.2

MANAGEMENT

OAIS Environment and Data Flows

Producer

Submission Information Packages

OAIS
Archival Information Packages

Consumer

queries

result sets

orders

Dissemination Information Packages

XML Formatted Data Units:Information Packages

XML Packaging Standard Rationale

Technology and Requirements Evolution

- Physical media -->Electronic Transfer
- No standard language for metadata--> XML
- Homogeneous Remote Procedure Call-->CORBA, SOAP
- Little understanding of long-term preservation-->OAIS RM
- Record formats-->Self describing data formats

New Requirements

- Describe multiple encodings of a data object
- Better describe the relationships among a set of data objects.

Technical Drivers

- ◆ Use of XML based technologies
  - ◆ Designed to be extensible to include new XML technologies as they emerge
- ◆ Linkage of data and software
- ◆ Direct mapping to OAIS Information Models
- ◆ Support both media and network exchange
- ◆ Support for multiple encoding/compression on individual objects or on entire package
- ◆ Mapping to current SFDU Packaging and Data Description Metadata where possible
- ◆ Maximal use of existing standards and tools from similar efforts

XML Packaging Efforts Studied

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

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

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.

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.

GLOBUS SERVICE TOOLKIT

The Grid Packaging Tool (GPT) uses XML to describe all the information that is needed for an installation.

XFDU Functionality by Version

Function/Feature	Version 1,0	Future Versions
Packaging techniques	1. Single XML Document 2. Archive File (e.g., tar, zip)	XML messaging form (e.g., Soap with Attachments)
Manifest	Mandatory	
Format Description Types	1. Markup Languages (XML and vocabularies) 2. MIME types, 3. Self describing formats 4. Detached data descriptions (e.g., EAST)	
Metadata/data linkage options	1.Inclusion in Manifest as base64 or XML, 2. Referenced directly as binary or XML 3. Referenced or Included as Data Object	
Relationship Description	1. Unit types indicate predefined relationships 2. Classification of metadata pointers 3. User defined metadata model support 4.Predefined support of OAIS Information model 5. Xlink attributes	Formal Description Language • RDF • OWL
Behaviors	1. Description of Abstract Interfaces 2. Inclusion of, or reference to, implementation specific mechanisms/methods	1. Invoking behavior as value of content units 2. Scripting Behaviors
Extensibility	Element substitution using XML Schema substitutionGroup, Redefine, XSD:any ,anyAttribute	1. Type Substitution using xsi: type
Encodings and Transformations	The ability to allow/reverse multiple transformations on files	
Instance Validation	XML schema type validation Enumerated lists	Constraints and business rules using Schematro, validate of XML data objects and MIME Type validation

Logical View of XFDU Manifest

DLF Fall 2006-Implementing OAIS Concepts
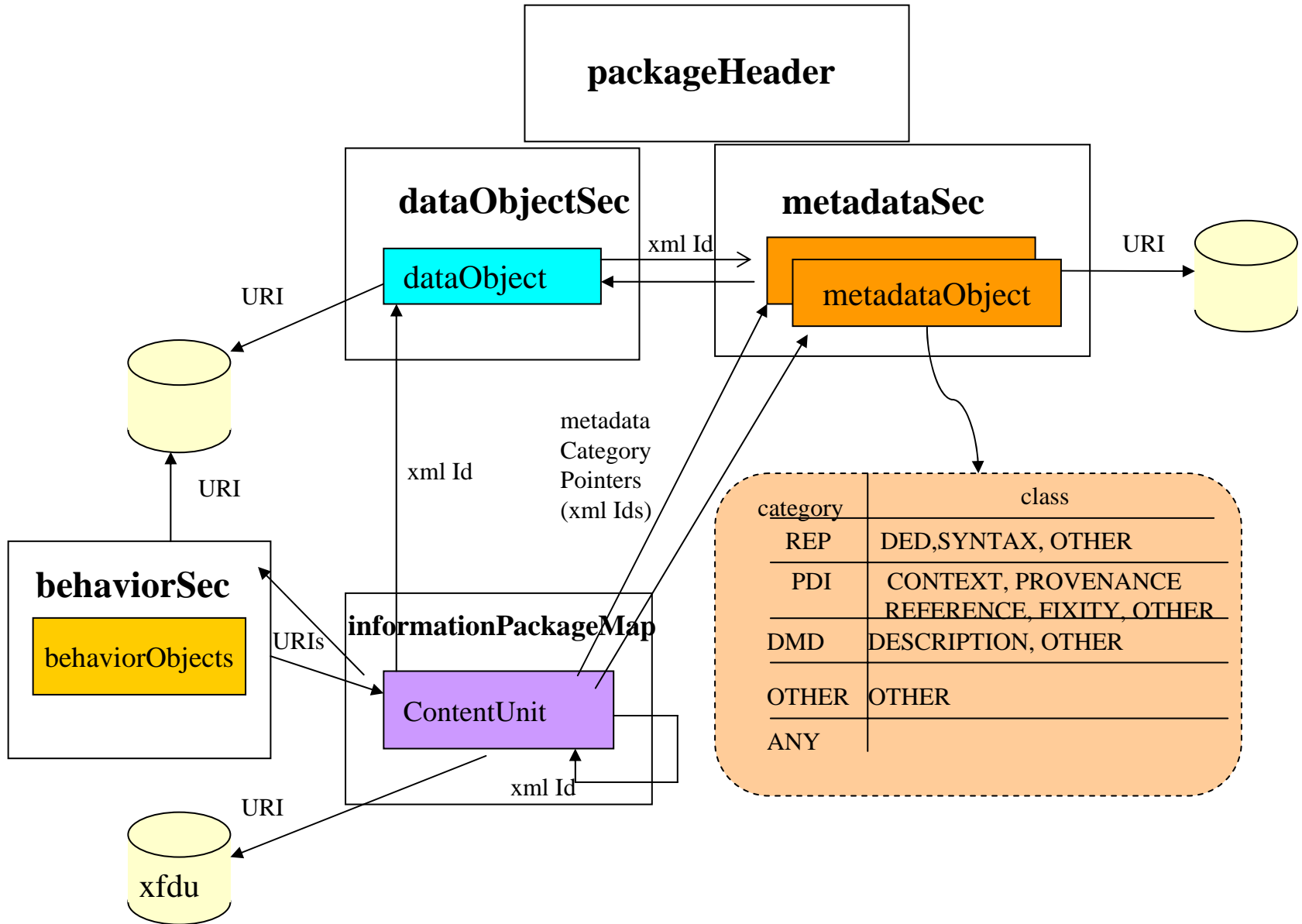
XFDU Reference Implementations

Development Approach

- Develop Draft Concept Paper and XML Schemas for internal review
- Use automated tool (JAXR) to develop JAVA Classes from XML schema
- Modify XML Schema based on internal review comments and issues based on JAVA class implementations
- Develop draft CCSDS White Book for Working Group Review
- Begin staged implementation of API layer and crude GUI of a packaging toolkit
  - **Toolkit should provide useful functionality at a very early stage for demonstration to interested parties**
- Present to Working Group for review and prototype commitments
- Develop specialization of schema that all international prototyping efforts agree to support
- Extensive usability and scalability testing in a variety of operational environments
- Revisions of the Schemas and Reference Implementation based on test results
- Develop new versions of XFDU Red Book and Tutorial based on Review Comments
- Develop XFDU/ Green Book tutorial/best practices based on testing experiences
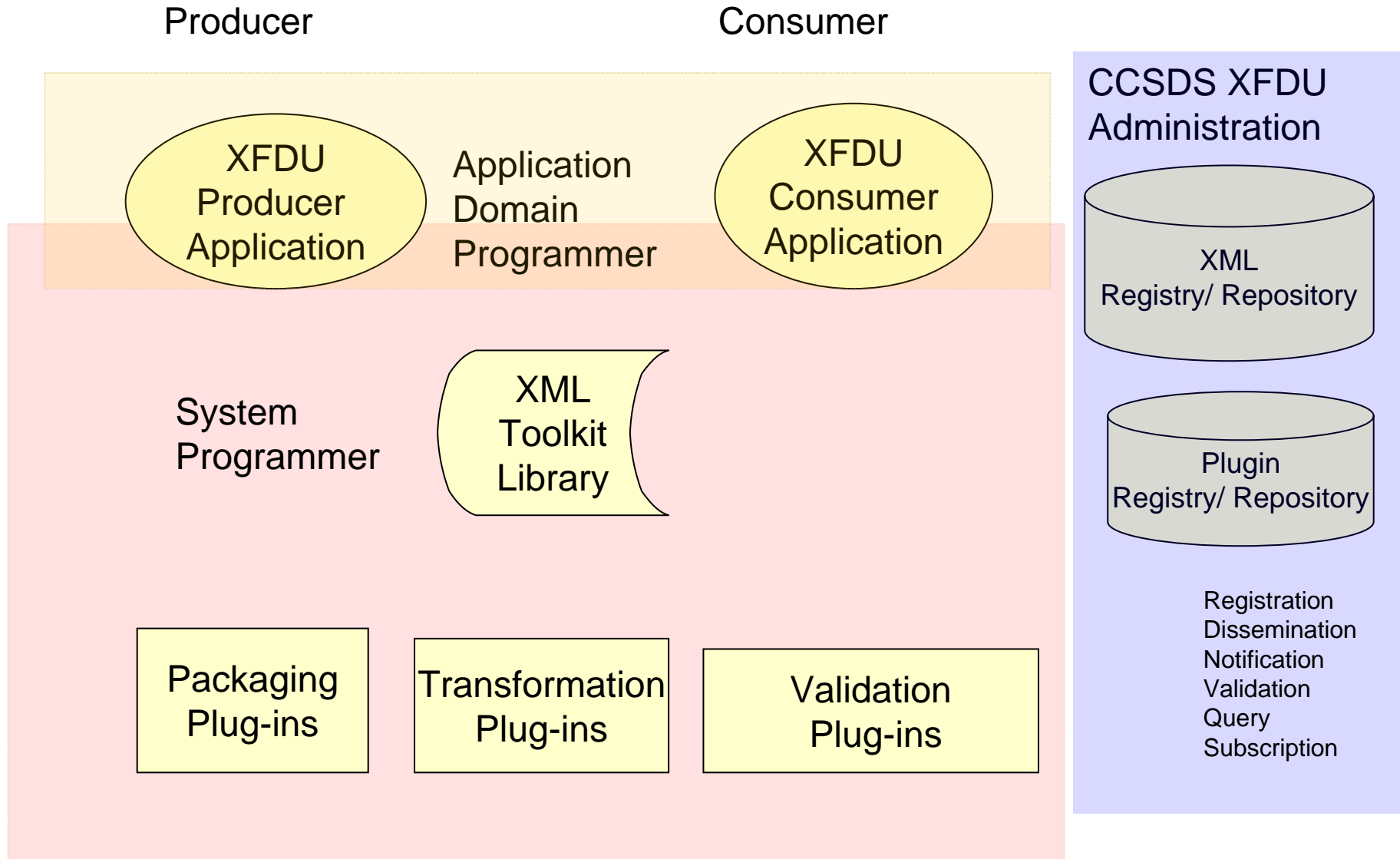
XFDU Toolkit Library

- Java class library implemented in parallel with specification development

- XFDU API library can be used within any other application that needs to perform XFDU packaging.

- To use XFDU toolkit in this manner simply put all the jar files inside of the lib directory on the CLASSPATH of your application. Javadoc documentation for the library can found at: http://sindbad.gsfc.nasa.gov/xfdu

- XFDU API Library conceptually consists of two layers

  - **First layer (xfdu.core.*)is low level API representing each structure in XFDU schema.**

  - **Second layer (xfdu.packaging.*) is more high level API. It aggregates parts of functionality from the first layer; thus, allowing easier access to constructing and manipulating an XFDU package.**

  - **A crude GUI is also supplied for testing and demonstration**

Conceptual XFDU Architecture

Producer

Consumer

CCSDS XFDU
Administration

XFDU
Producer
Application

Application
Domain
Programmer

XFDU
Consumer
Application

XML
Registry/ Repository

System
Programmer

XML
Toolkit
Library

Plugin
Registry/ Repository

Packaging
Plug-ins

Transformation
Plug-ins

Validation
Plug-ins

Registration
Dissemination
Notification
Validation
Query
Subscription

Current NARA Joint Research Results and Future Plans

DLF Fall 2006-Implementing OAIS Concepts

Implementing Advanced XFDU Functionality

Test Design

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

Implementing Advanced XFDU Functionality-2

Test Observations and Conclusions

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

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

DLF Fall 2006-Implementing OAIS Concepts

Implementing Advanced XFDU Functionality-3

Scalability Testing

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

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

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

Implementing Advanced XFDU Functionality-4

Test Results and Observation

Validation via XSD

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

Schematron validation

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

Implementing Advanced XFDU Functionality-5

Issues and Conclusions

- The validation of individual data objects appears to be very scalable through 5000 objects. The observation that the time/object decreases as the number of objects increase is probably due to a fixed or slowly increasing initial overhead that is being distributed over an increasing number of objects.
- The Schematron mechanism does scale very well.
    - Given that Schematron uses XSLT and XPATH, the problem could be similar to the JXPATH issue we encountered in the XFDU creation tests.
    - However, we didn't implement Schematron so it is doubtful we can optimize it.
- We will need to include manifest size limits for the use of Schematron to validate XFDU rules in XFDU Best Practices
- Clearly, many more scalability test are needed however these tests indicate that using XSD validation on individual data objects appears to be a viable mechanism for allowing third party extensions that are independent of other extensions.

Summary: XFDU Findings

- The status of XML schema versioning and extensibility mechanisms is clearly a concern in the development of the XFDU and in the use of XML, described by XML Schema, as an Archival Format. This finding was a conclusion of the XML Schema Specialization Best Practices Study and was confirmed by the practical efforts to extend the XFDU XML schema to create the SIP mechanisms.

- The mechanism of validating individual data objects using the Sun Multi-schema Validator through the XFDU toolkit validation interface provides significant added value to the XFDU Toolkit Library and may provide a practical solution to the unresolved issue of extending the XFDU XML Schema by third parties. Preliminary testing of the current implementation has shown this mechanism to be highly scalable.

- The approach of developing one or more high quality reference implementations in parallel to the specification has proven very valuable in identifying unclear portions of the specification and building confidence of potential users. This was demonstrated by the NSSDC/PDS test-bed experience and commitment to expanding the scope of the test-bed.

- The modeling of the USGS/NARA products confirmed the flexibility of the XFDU information model and the ability to easily express OAIS Information Model Representation Networks

XFDU Current Research Issues

- Definition of mechanisms to enable effective implementation of behaviors and compact definition of relationships for XFDU version 2.

- There is a requirement for much more performance and scalability testing using anticipated data loads for NARA ERA As a prerequisite for this effort the XFDU Toolkit library will need to be migrated to a 64 bit environment

- There needs to be much more testing of the use XFDU structures using real existing data

- There needs to be a study on the interoperation or at least the co-existence of the XFDU and METS and best practices document to assist in the selection

- The study of XML Specialization and Validation should be extended an include the definition of an Archival Profile of XML Schema and the potential use of of ISO/IEC 19757 Document Schema Definition Languages (DSDL) languages as an alternative to XML schema

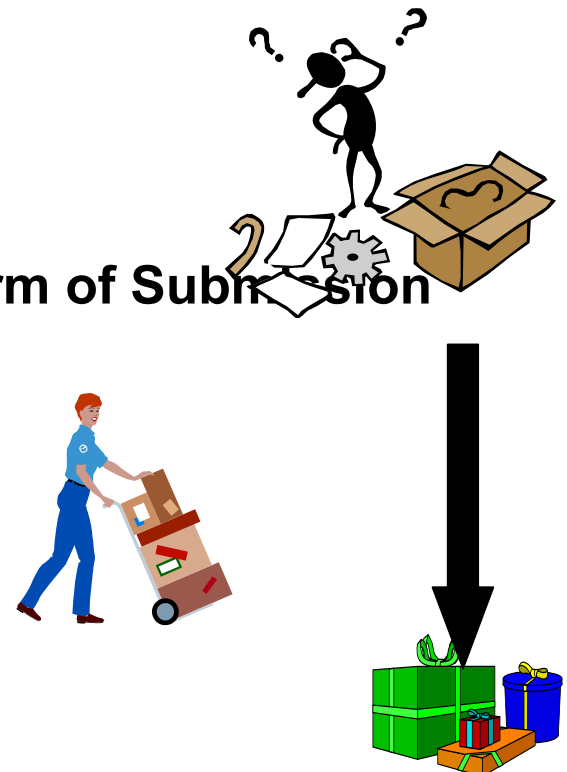# *Producer Archive Interface Specification*

Objectives of Standard
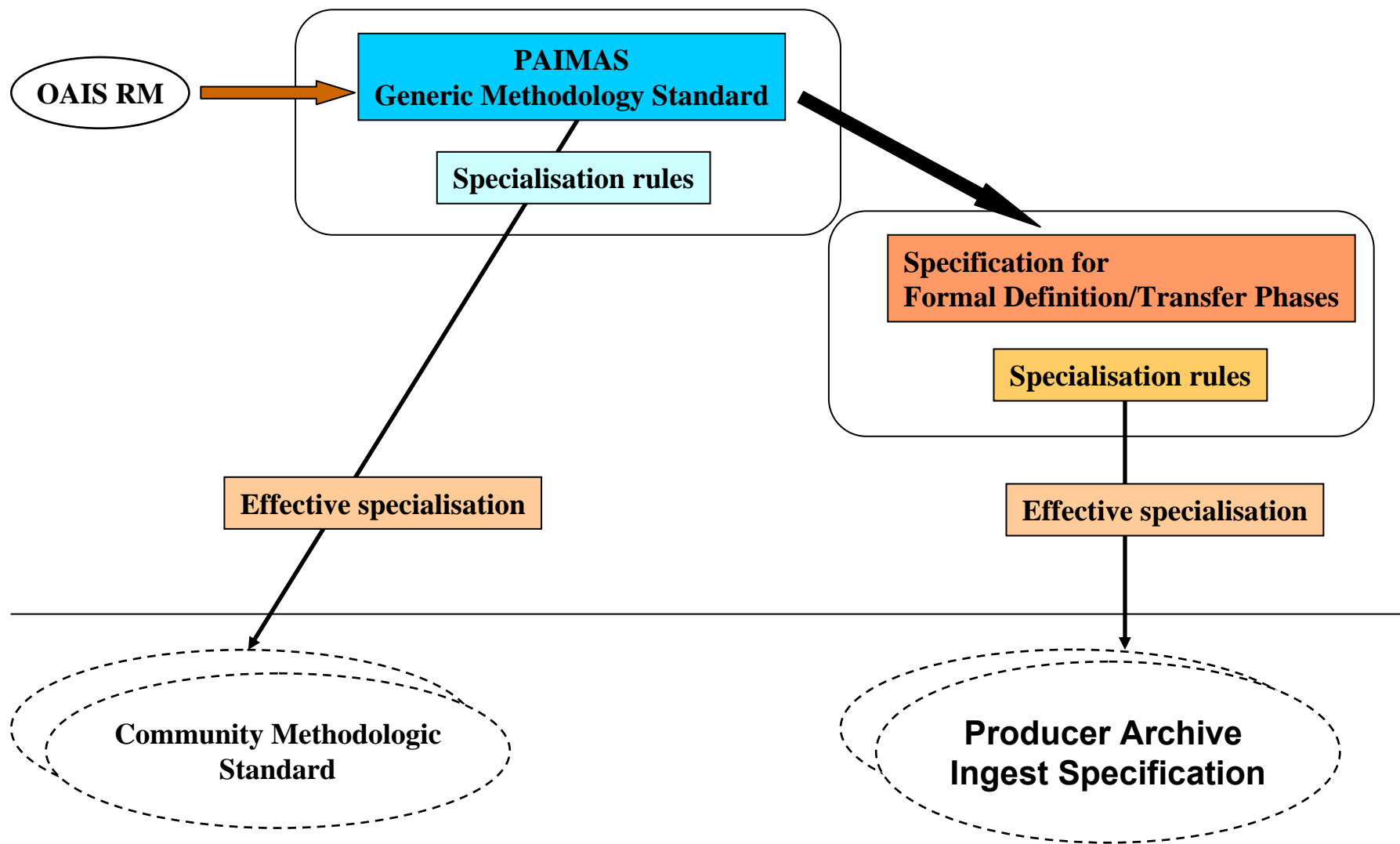
Provide a standard method to

formally define the digital information objects to be transferred by a Producer to an Archive

effectively transfer these objects in the form of Submission Information Packages (SIPs)
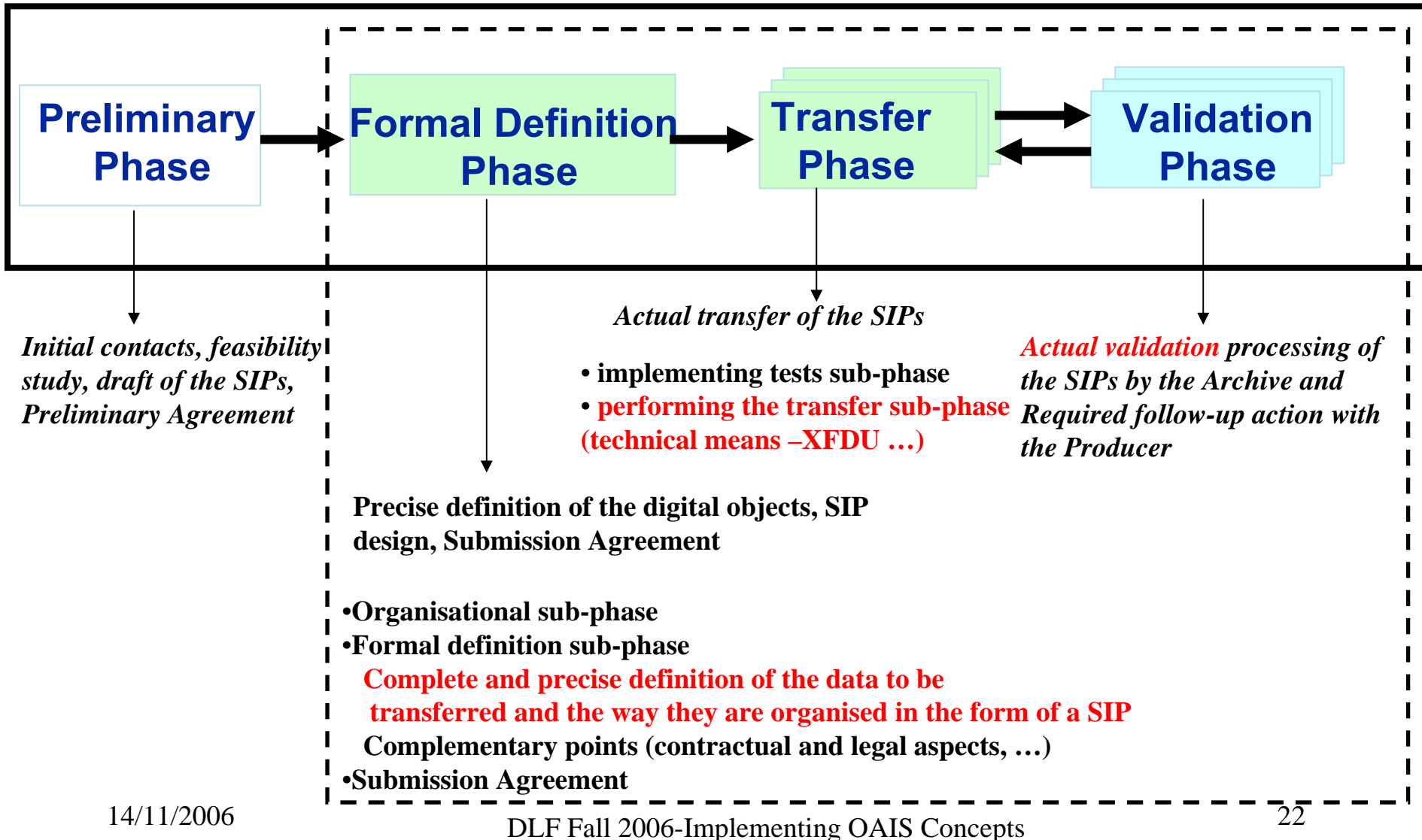
validate the SIPs

DLF Fall 2006-Implementing OAIS Concepts

Evolution of Data Archive Ingest standards

OAIS RM

PAIMAS
Generic Methodology Standard

Specialisation rules

Specification for
Formal Definition/Transfer Phases

Specialisation rules

Effective specialisation

Effective specialisation

Community Methodologic
Standard

Producer Archive
Ingest Specification

PAIS Support in Context of PAIMAS

Preliminary Phase → **Formal Definition Phase** → **Transfer Phase** ⇄ **Validation Phase**

Initial contacts, feasibility study, draft of the SIPs, Preliminary Agreement

Actual transfer of the SIPs

- implementing tests sub-phase
- **performing the transfer sub-phase (technical means –XFDU …)**

Actual validation processing of the SIPs by the Archive and Required follow-up action with the Producer

Precise definition of the digital objects, SIP design, Submission Agreement

- Organisational sub-phase
- Formal definition sub-phase
  Complete and precise definition of the data to be transferred and the way they are organised in the form of a SIP
  Complementary points (contractual and legal aspects, …)
- Submission Agreement

DLF Fall 2006-Implementing OAIS Concepts

XML Based Standardization of Descriptions of Data to be Transferred
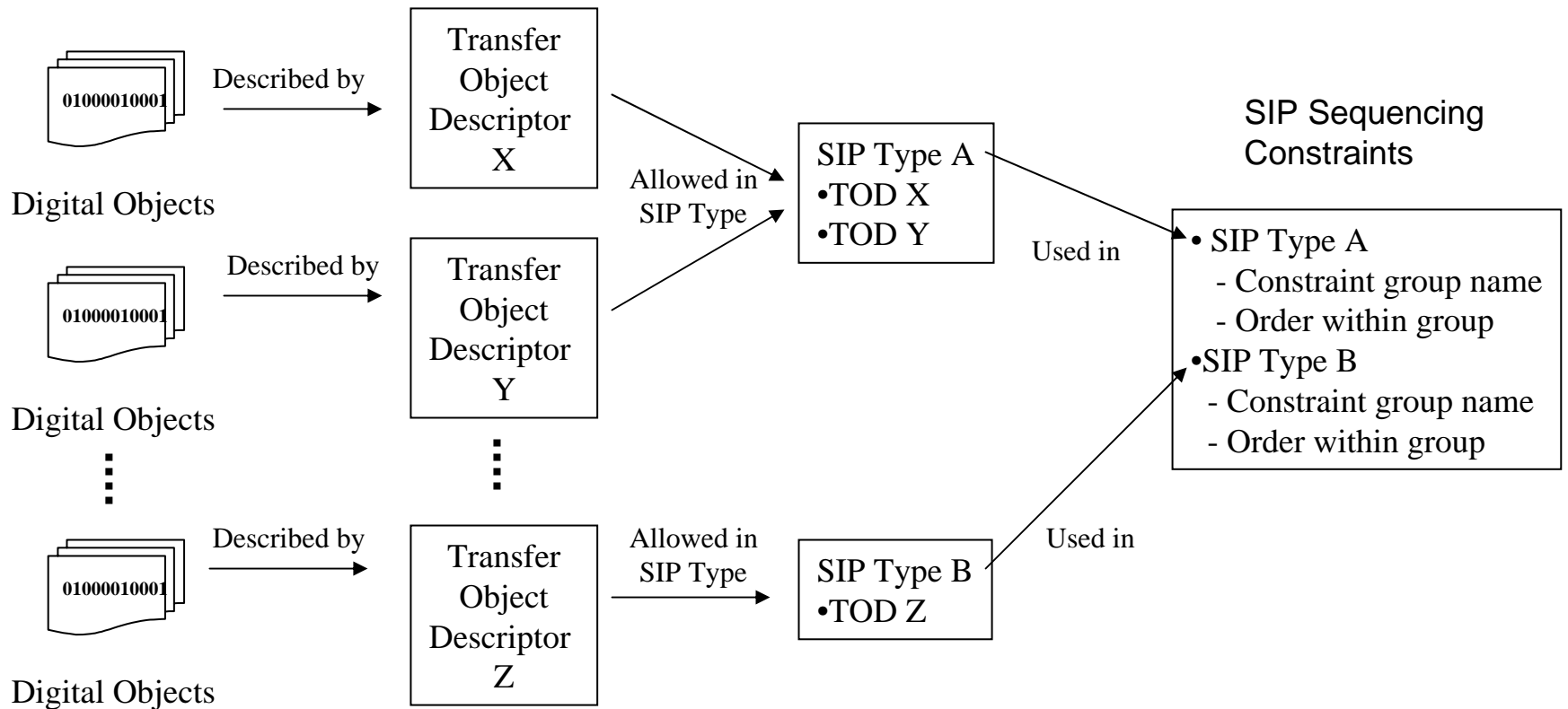
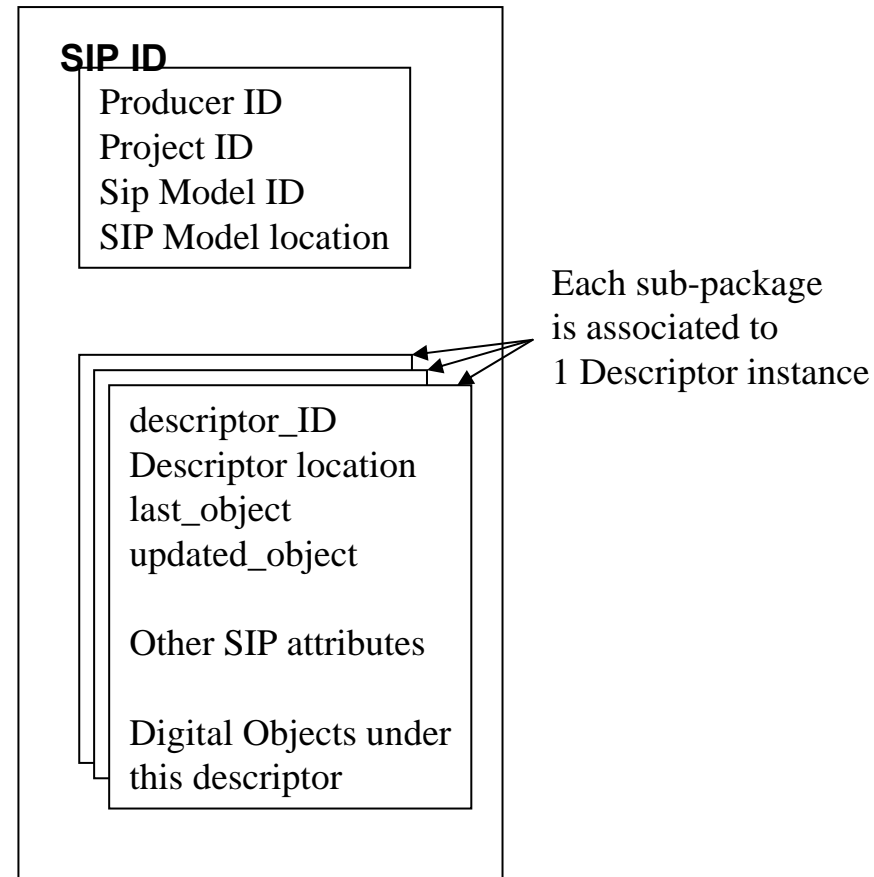| Data to be transferred | Descriptors are attributes describing groups of data to be transferred (XML Descriptors) | SIPs are categorized into SIP Types constrained to only contain identified Transfer Object Descriptors (XML Grouping Constraints) | SIP Types may have constraints on the sequence of their transfer (XML Sequencing Constraints) |

SIP Abstract Model

◆Global information (M, 1..1)

◆Information associated with each embedded object (M, 1..N)

◆Other information (M, 1..1)

◆Digital Object (O,0..N)

SIP ID

Producer ID
Project ID
Sip Model ID
SIP Model location

Each sub-package
is associated to
1 Descriptor instance

descriptor_ID
Descriptor location
last_object
updated_object

Other SIP attributes

Digital Objects under
this descriptor

SIP Implementation as XFDU

Initial Proposal

    Map SIP global attributes to XFDU P_a Header

    Map SIP descriptor ID and related attributes to XFDU Content Units and their attributes

PAIS Workplan Thru January 2007

- Develop test cases and apply the draft PAIS techniques
- Develop SIP to XFDU mapping proposals
- Analyze test cases and identify needed updates
- Produce new draft standard (CNES)

OAIS Reference Model Review and Update

Current Work Plan for OAIS Reference Model Review

It has been agreed that we should not try to drill down into more detail; we should try to identify gaps at the current general level of detail of the OAIS Reference Model.
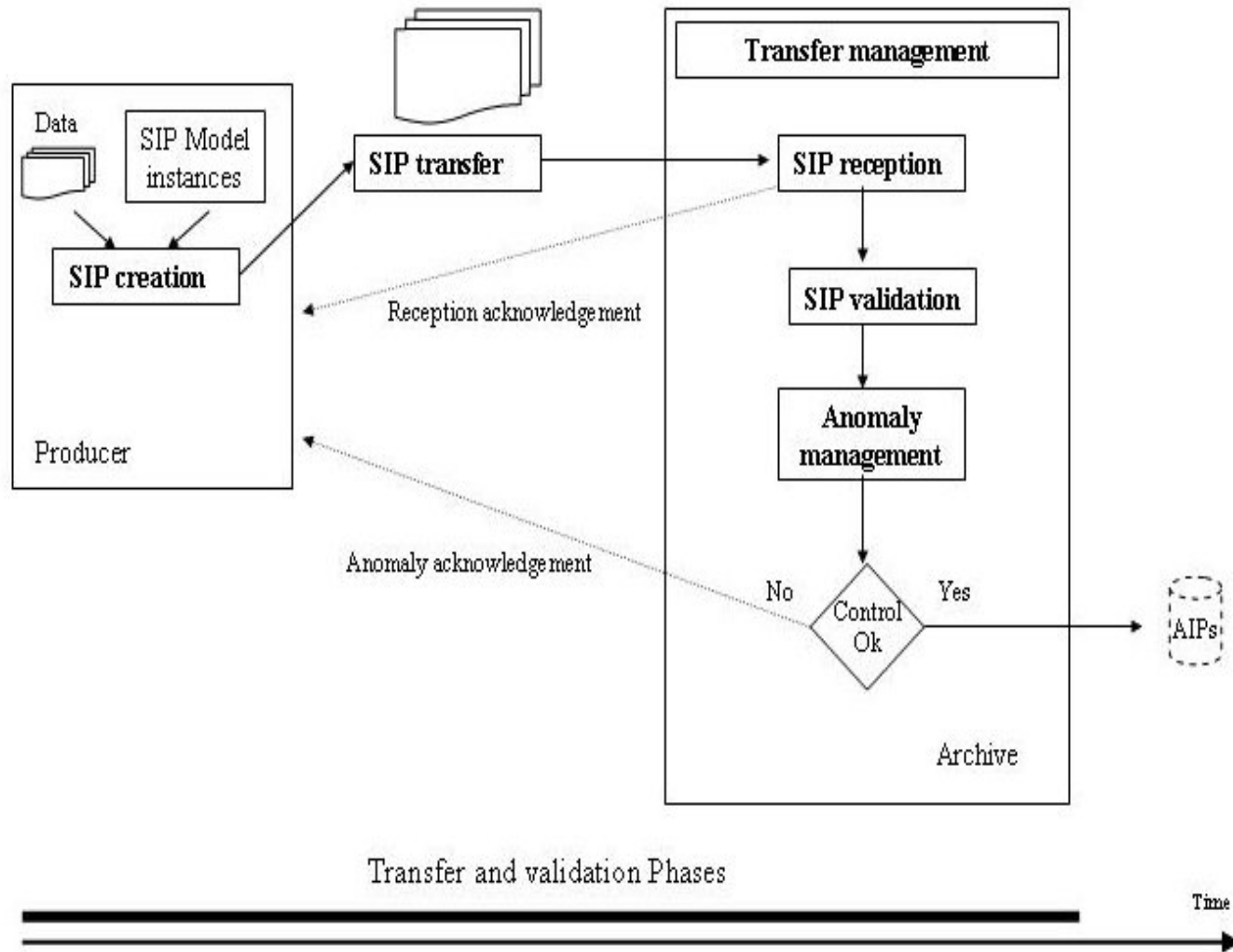
- **End July 2006: Draft a Notice, with some description of constraints, on the intent to update the OAIS RM per the 5-year review cycle coming due January 2007. .**

- **End September 2006: Provide Agencies with revised Notice and POCs recommended for each agency's distribution. This will include newsletter, etc. with instruction widely. This should include ISO newsletter or bulletin board. Agencies send notice to Points of Contact. Require responses by 30 October to influence establishment of the update scope.**

- **End November 2006: Analyze and sort responses**

- **January 2007: Begin deliberations on what to address, what to reject. Also consider soliciting an organization to host an "OAIS usage experience" workshop.**
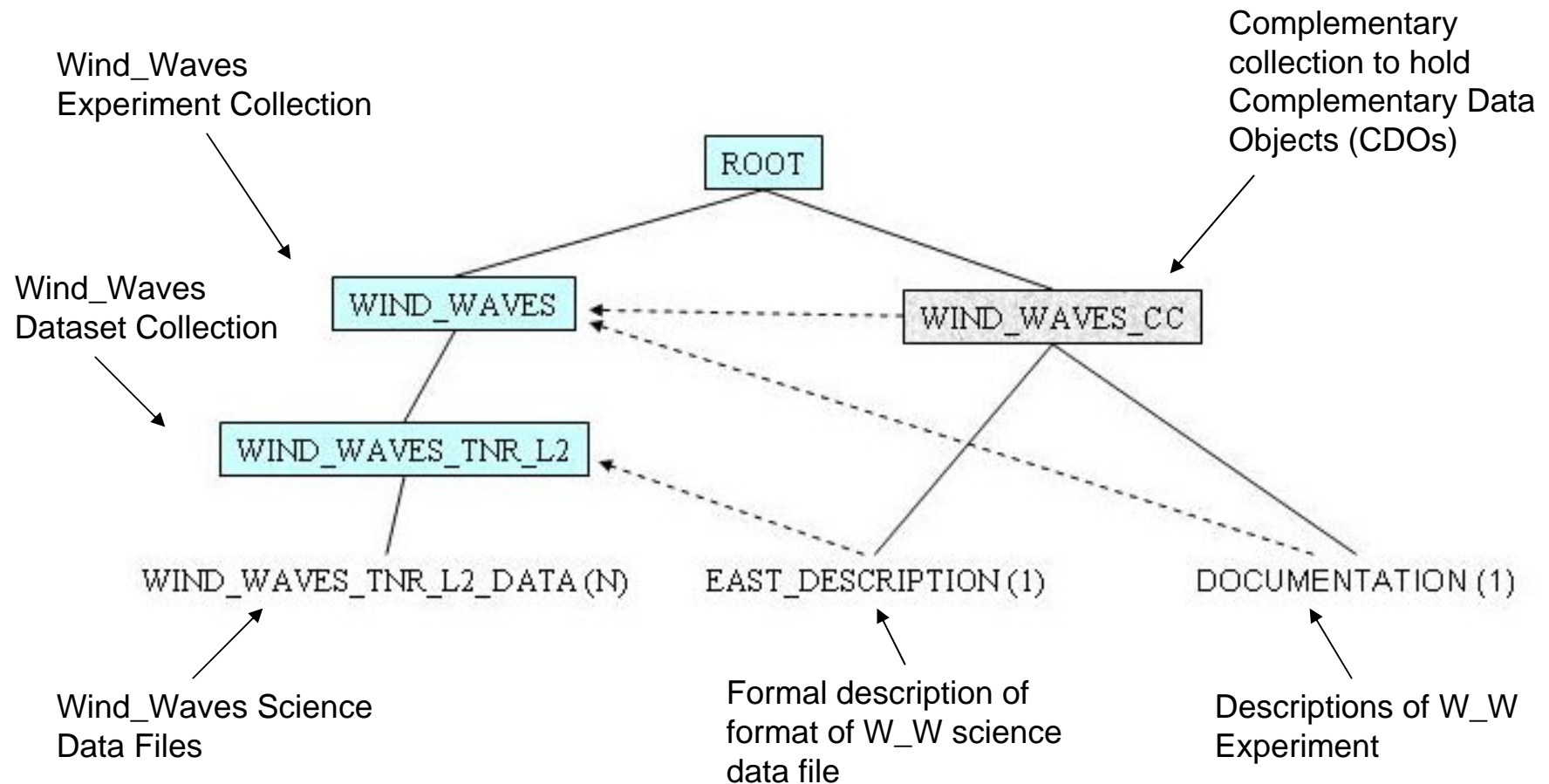
BACKUPs

Transfer and validation Phases
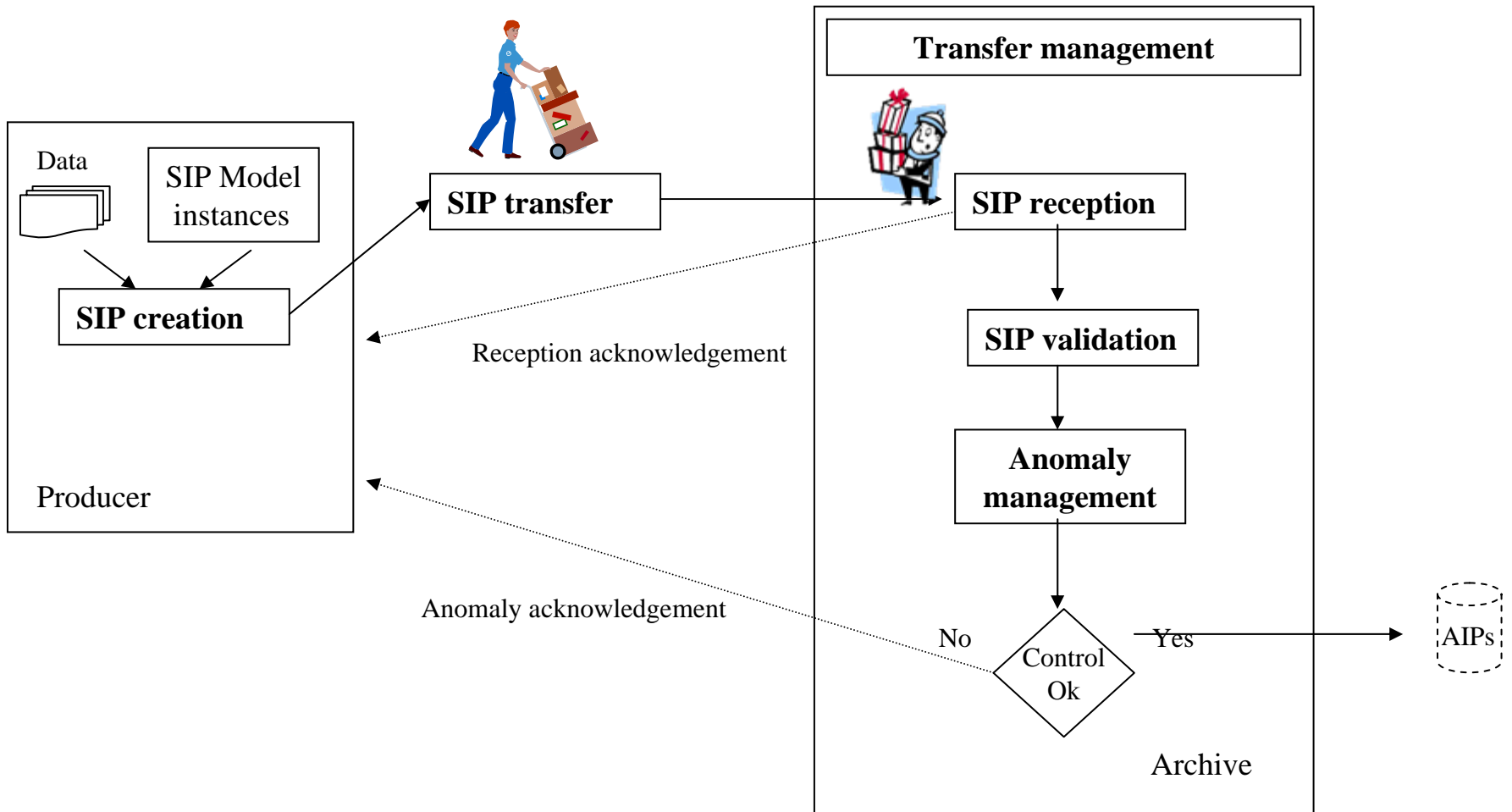
Time

General Process: SIP Creation to SIP Validation

Example Modeling of Collections to Transfer: Object Content Hierarchy

Wind_Waves
Experiment Collection

Wind_Waves
Dataset Collection

Complementary
collection to hold
Complementary Data
Objects (CDOs)

ROOT

WIND_WAVES

WIND_WAVES_CC

WIND_WAVES_TNR_L2

WIND_WAVES_TNR_L2_DATA(N)

EAST_DESCRIPTION(1)

DOCUMENTATION(1)

Wind_Waves Science
Data Files

Formal description of
format of W_W science
data file

Descriptions of W_W
Experiment

Nodes and Descriptors

- Each Node (except Root) of the object content hierarchy is described by a Descriptor Object
  - Descriptors are specialized from those defined in the Standard (currently DO, CDO, Collection of DO, CDO)
- Set of Descriptors provides POT information
  - Agreed between Producer and Archive prior to SIP transfers
- Leaf nodes are packaged in SIPs for transfer to archive
  - SIP Models/template include Descriptor Identification
  - Each Descriptor identifies type of SIP template to be used for the associated data objects to be transferred

Example Descriptor for a CDO

descriptor_type	DOCUMENTATION
descriptor_ID	WAVES_DOCUMENTATION
object_occurrence	1..1
title	WAVES: The Radio and Plasma Investigation on the WIND spacecraft
content	WIND XAVES mission description
parent_collection	WIND_WAVES_CC
related_descriptor_ID	WIND_WAVES
relation_with_DO	Experimental textual description
metadata_schema	doc_waves.xsd
SIP descriptor ID	EXPERIMENT_DESCRIPTION_SIP

This part contains everything that is known about the object at the time of the Formal Definition Phase (characteristics of the object and its relations to other objects)

The SIP descriptor describes the organisation of the Unit Package

Descriptor Overview

● **Descriptor Model**, attributes for:

  ⊞ Descriptor identification: identifying the type of object described.

  ⊞ Content description: describing the object content.

  ⊞ Relationships within the POT: defining the relations between the object described and the other Model objects.

  ⊞ Object content: describes all the Data and Metadata included in this Descriptor.

  ⊞ Specialisation section.

  ⊞ SIP Model reference.

(Note: Annex A.1 of draft standard provides XML schema for the Generic Descriptor)

Identification

● **Identification** (M, 1..1)

- ⊞ **descriptor_type (M, 1..1)**: defines the type of Object Model and is constant for a given type of Descriptor.

- ⊞ **descriptor_ID (M, 1..1)**: Descriptor Identifier, used to identify the nodes in the POT.

- ⊞ **object_occurrence** (**M, 1..1**): number of Data Objects described by this descriptor_ID. This Descriptor instance may describe a unique Object (this is often the case for a Data Collection, or a syntactical description of a Data Collection), or a representative of a category containing a certain number of Objects. This number may not be known ahead of time (value 1..n).

- ⊞ **version (M, 1..1)**.

● **Content description** (M, 1..1)

- ▣ **title (M, 1..1):** extensive name of the Object.
- ▣ **content (M, 1..1):** explanatory text specifying the content and main characteristics of the described Object.
- ■ **size (O, 0..1):** estimated volume of an object and unit.
- ■ **Non definite attribute (O, 0..1):** specialization attribute (to be defined if necessary). The name and form of this attribute depends on the future implementation.

Relationships

● **Relationships within the POT (M, 1..1)**

- ⊕ **parent_collection (M, 1..1)**: identifier of Collection Descriptor to which this object belongs.
- ■ **association (O, 0..N)**: this is used when the Descriptor describes a Complementary Data Object or a Complementary Collection (transversal links).
  - **related_descriptor_ID (M, 1..1) :** Descriptor identifier for the Data Object Collection or Data Object to which this Object refers
  - **relation_with_DO (O, 0..1):** type of relation between the Object in question and the related_descriptor_ID.

Object Content

● **Object content (O, 0..N).**

⊞ **content_ID (O, 0..1):** the Data section describes the Data Object itself, while the Metadata section describes the Metadata attached to the Data.

⊞ **data (O, 0..1):**
  - **data_ID (M, 1..1):** Data Identifier.
  - **data_content (O, 0..1):** Explanatory text specifying the content and main characteristics of the described Data
  - **data_format (O, 0..1):** Format of objects in the collection (PNG, PDF, CDF, Flat Binary, Flat ASCII, …)

⊞ **Metadata (O, 0..1)**
  - **metadata_ID (M, 1..1):** Metadata Identifier.
  - **metadata_content (O, 0..1):** Explanatory text specifying the content and main characteristics of the described Metadata.
  - **metadata_format (O, 0..1):** Used to specify the metadata standard to be applied.
  - **metadata_schema (O, 0..1):** Used to specify the 'catalogue' data characterising the object: for instance in the form of an XML document specified by an XML Schema.

User Defined and SIP Reference Sections

- **Non definite section:** optional section in which can be included any attributes to specialise the Descriptor.

- **SIP Model reference:** SIP Model Identifier. For a collection of collections, there may not be a corresponding delivery during the Transfer Phase, and hence no applicable SIP Model: the attribute value is then NONE.

Linking Descriptors and SIPs
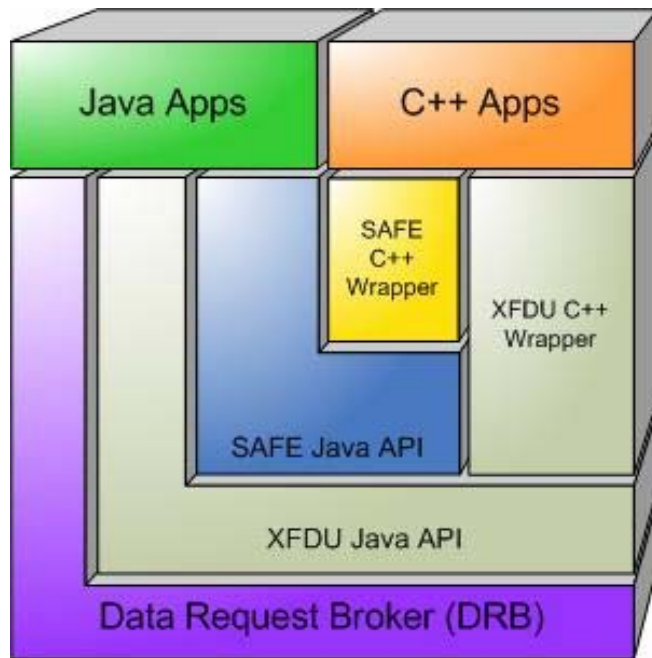
DLF Fall 2006-Implementing OAIS Concepts
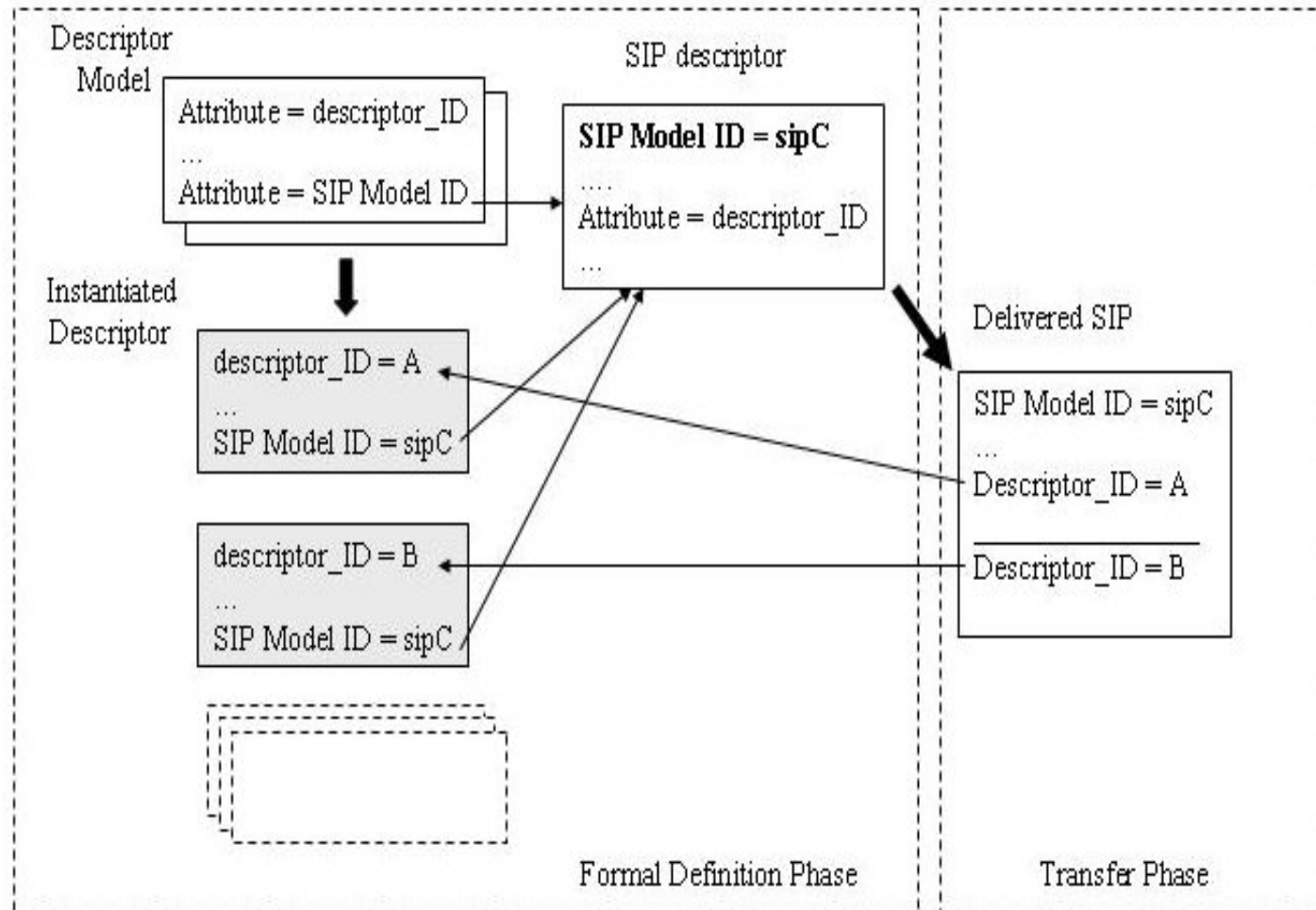
SIPs

- Abstract SIP model has attributes defined in the standard (like Descriptor)
- SIPs may have sequencing constraints (one before the other in the transfer)
- SIPs will be contained in XFDUs, but precise mapping is TBD
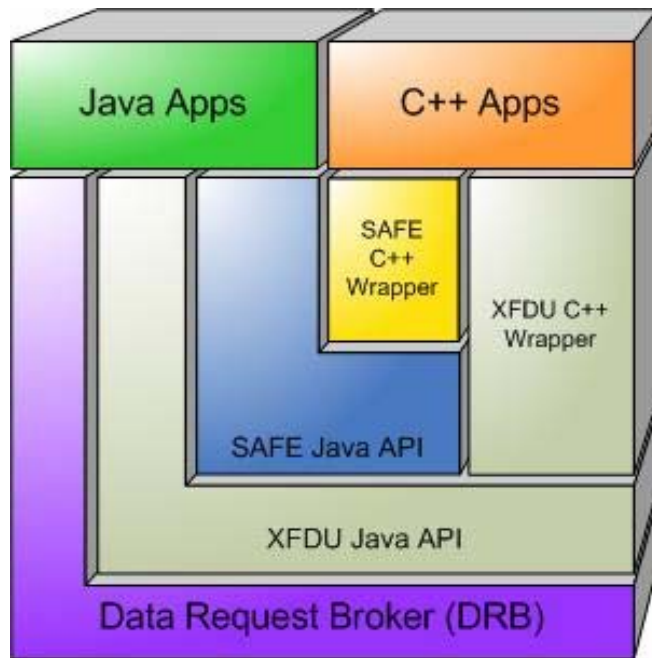
ESA/CNES SAFE I/O API

- **XFDU Java API**: this API provides the general features common to all XFDU packages. This API has been developed jointly with the SAFE Java API and is presently used in the framework of the activity of the CCSDS Information Packaging and Registries (IPR) Working Group in support to the validation of the XFDU recommendation developed by the group
- **XFDU C++ Wrapper**: a C++ wrapper on top of the XFDU Java API. Most of the functionality is preserved from the Java API, apart the capability of browsing the binary content
- language used for annotating the XML Schema documents acting as representation information of the SAFE product objects.

ESA/CNES SAFE I/O API

- **XFDU Java API**: this API provides the general features common to all XFDU packages. This API has been developed jointly with the SAFE Java API and is presently used in the framework of the activity of the CCSDS Information Packaging and Registries (IPR) Working Group in support to the validation of the XFDU recommendation developed by the group
- **XFDU C++ Wrapper**: a C++ wrapper on top of the XFDU Java API. Most of the functionality is preserved from the Java API, apart the capability of browsing the binary content
- language used for annotating the XML Schema documents acting as representation information of the SAFE product objects.

Technical Detail: Information Package Concepts
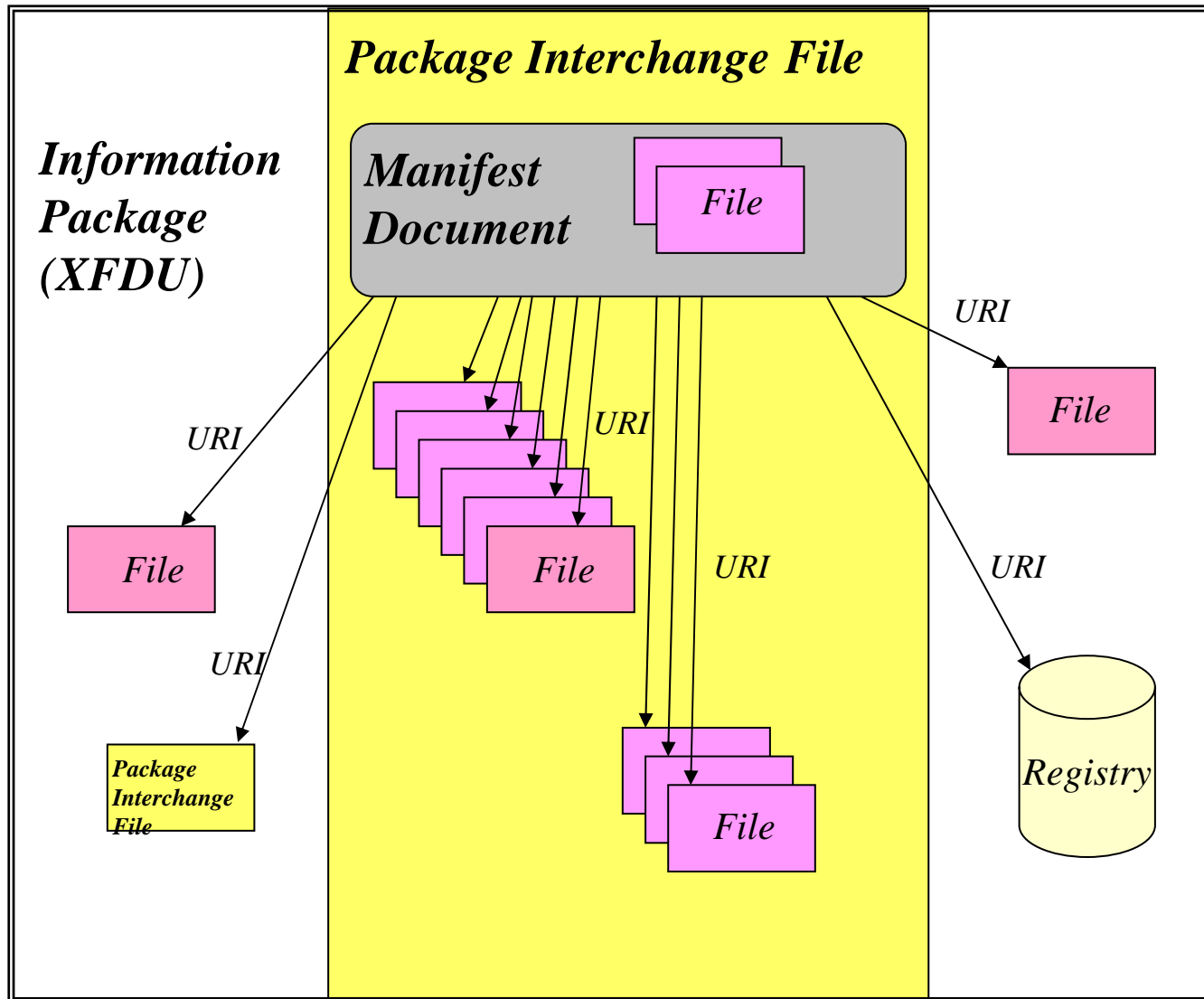
XFDU Toolkit Library Performance Testing

Environment

All test were performed using the following:

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

Use Case Description

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

XFDU Toolkit Library Performance Testing-2

Test Descriptions

1. Create an XFDU package in ZIP format from a directory structure with 31000 files
   a. Using XFDU APIs, a package is created both with and without checksum computation and saved in ZIP format.
   b. Using XFDU APIs, the manifest is extracted from the package.
   c. Using XFDU APIs, a randomly selected dataObject (file) is extracted from the package.
   d. Using XFDU APIs, the package is opened and the files are expanded optionally validating any checksums recorded in the XFDU Manifest object.
2. Create an XFDU package in ZIP format out of a directory structure with 2 DivX compressed files
   e. Steps a-d above
3-4. Repeat tests 1 and 2 using the JAR binary archive format
5-6. Repeat tests 1 and 2 using the TAR/GZIP binary archive format

XFDU Toolkit Library Performance Testing-3

Initial Results

- It was observed that package creation of the 31000 object XFDU using the XFDU API did not complete after 12 hours. Upon closer investigation, it became obvious that the slowness could be attributed to usage of the JAVA implementation of XPath while constructing the Java Object tree.

- The observed behavior (in regards to using XPath) only become noticeable when thousands of files are being packaged which results in an XFDU manifest of significant size.

- As a result, that part of the XFDU API was completely rewritten to perform necessary lookups using only memory object references. This improved performance significantly (from hours to minutes).

XFDU Toolkit Library Performance Testing-3

Initial Results

- It was observed that package creation of the 31000 object XFDU using the XFDU API did not complete after 12 hours. Upon closer investigation, it became obvious that the slowness could be attributed to usage of the JAVA implementation of XPath while constructing the Java Object tree.

- The observed behavior (in regards to using XPath) only become noticeable when thousands of files are being packaged which results in an XFDU manifest of significant size.

- As a result, that part of the XFDU API was completely rewritten to perform necessary lookups using only memory object references. This improved performance significantly (from hours to minutes).

XFDU Toolkit Library Performance Testing-4

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

XFDU Toolkit Library Performance Testing-5

Observations

- The time required for packaging and unpacking the test XFDUs using the XFDU Toolkit interface was two-three times longer than the time needed to create packages in the previous test of the native interfaces. However the major reason appears to be the performance of the JDK implementations of the compression methods rather than added processing needed to create XFDU structures.

- The TAR/GZIP tests confirm the conclusions in the previous section that the extraction of the manifest and extracting specific files will not work unless the underlying package has indices to enable efficient random access and extraction of single objects

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

- Use Case 1: Transfer of Planetary Data System (PDS) Volume to NSSDC
- Use Case 2: Packaging USGS Data Submitted to NARA