Recommendations for Preservation-aware Digital Object Model

Authors
Matthias Hahn (Fachinformationszentrum Karlsruhe)

January 2014

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

Executive Summary

The SCAPE project deals not only with diverse scenarios, file formats and complexity of objects but also with different digital object repositories. The digital object repositories like RODA, Rosetta and eSciDoc have to be integrated into the SCAPE environment and several APIs have been defined. But defining APIs is only one part of the challenge. The other part is an object model that can be understood by every repository that likes to share information between each other or with the SCAPE environment. This report describes recommendations for an object model for large-scale repositories dealing with long-term preservation of digital content. Starting from the OAIS Reference model along with METS and PREMIS in general we focus on existing repositories and how they deal with digital objects. As a result we present the SCAPE Digital Object Model, the implementation of this model and the Data Connector API. For a Digital Object Repository that wants to integrate with the SCAPE platform it is recommended to implement the SCAPE Digital Object Model and the Data Connector API on top of the repository.

Table of Contents

1 Introduction

The SCAPE project deals not only with diverse scenarios, file formats and complexity of objects but also with different digital object repositories. The DORs like RODA, Rosetta, eSciDoc have to be integrated into the SCAPE environment and several APIs have been defined. But defining APIs is only part of the challenge. To be able to share information between the preservation tasks performed on an executable platform the repository must implement an API and an object model that allows the interchange of information. The DOR partners of SCAPE (KEEPS, FIZ, and ExLibris) agreed on a Digital Object Model within the SCAPE project. While it is obvious that each repository already provides a Digital Object Model, the diversity hinders the ability of the SCAPE platform to integrate all the partner repositories. This Document resolves this issue and provides recommendation for an object model and a Data Connector API for Digital Object Repositories.

The following high-level overview of the SCAPE architecture illustrates the integration of the DOR in the SCAPE environment.

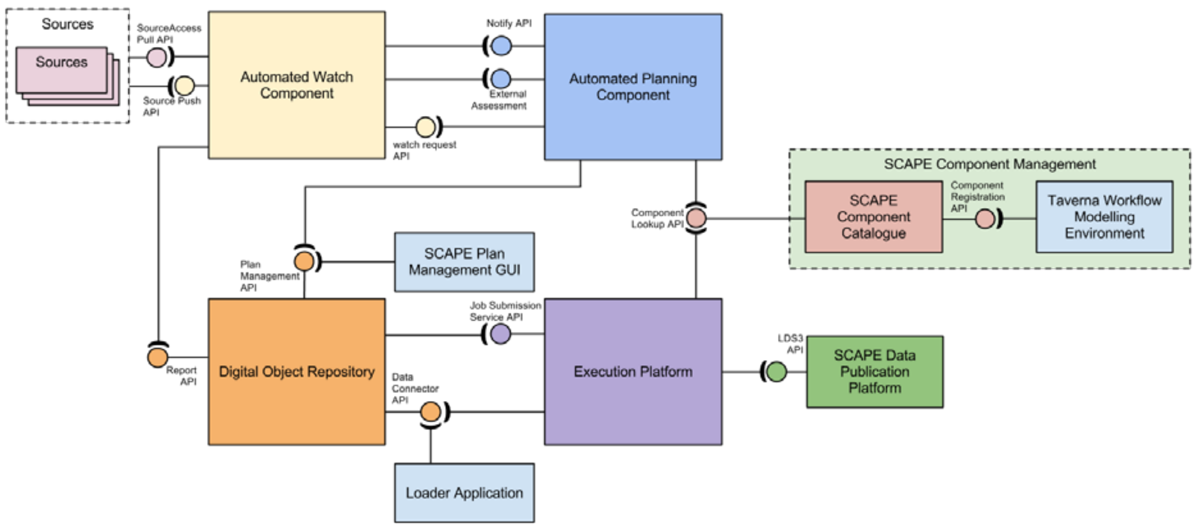

Sources
Sources
SourceAccess Pull API
Source Push API
Automated Watch Component
Notify API
External Assessment
watch request API
Automated Planning Component
Component Lookup API
SCAPE Component Management
SCAPE Component Catalogue
Component Registration API
Taverna Workflow Modelling Environment
Plan Management API
SCAPE Plan Management GUI
Job Submission Service API
Report API
Digital Object Repository
Data Connector API
Execution Platform
LDS3 API
SCAPE Data Publication Platform
Loader Application

Figure 1-1: SCAPE Architecture overview

In this document we are going to focus on the integration of the DOR with the Execution platform, because this integration is central for ingest, retrieve and update the digital content of the repository.

In order to use the same terminology throughout the document we give a short introduction into the well-known standards like OAIS, METS and PREMIS used in the long-term preservation world. Some questions related to our domain specific requirements are discussed.

The reference model for archives, the OAIS model, will be discussed in brief on an abstract level. The METS standard describes a XML container for metadata structure of digital objects. METS is widely used for interoperability between repositories and service components. The PREMIS standard describes a semantic model for preservation metadata, and is widely used in long-term preservation.

Using METS and PREMIS together will be discussed briefly since there are some issue one has to be aware of.

Some of the repositories of the SCAPE members already use METS and PREMIS, but the mere employment of these standards does not guarantee interchangeability between digital repositories. There might for example be significant differences in between two METS documents as used by two different repositories. We will describe the current existing data models of the repositories in this document and develop a possible model every repository holder may subscribe to.

The outline of the document is as follows: after a short introduction into the Open Archival System (OAIS), METS and PREMIS we are going to discuss the current digital object model of each repository of SCAPE. The next section will describe the Digital Object Model that will be used within the SCAPE project. Last but not least we'll define an API that is able to consume the defined Digital Object Model.

2 Theoretical Background

This chapter deals with some basic understanding of wide spread standards and de-facto standards used in long term preservation projects. We start with most abstract description of a repository for long-term preservation named Open Archival System. The purpose is to introduce the reader into the naming convention that we will use in this document. An introduction into METS and PREMIS and the interplay of both standards is given as well in this chapter. Understanding METS and PREMIS is crucial to understand the SCAPE Digital Object Model, because it builds on these standards.

2.1 OAIS

OAIS is the acronym for an Open Archival System and describes on an abstract level the requirements an archival system for long-term preservation has to fulfil. The reference model describes the following six functional areas:

1. Ingest
2. Archival Storage
3. Data Management
4. Administration
5. Preservation Planning
6. Access

The key terms for this document are SIP (Submission Information Package), AIP (Archival Information Package) and DIP (Dissemination Information Package).

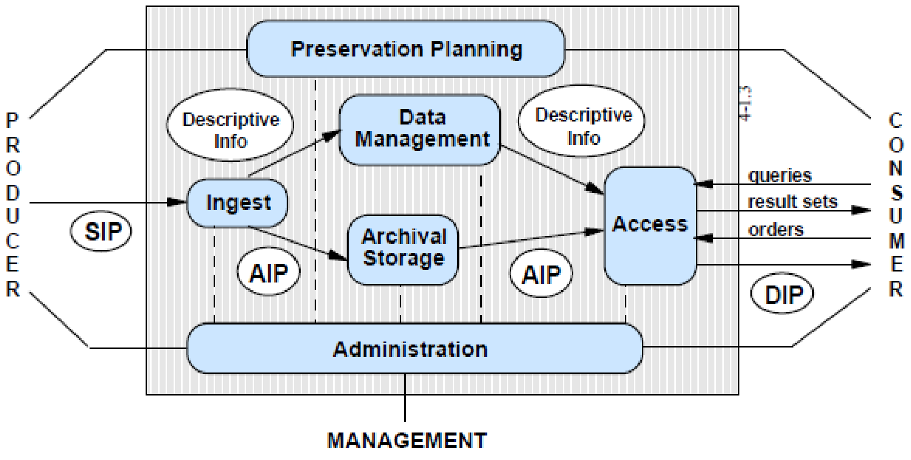


Figure 2-1: Illustration of the OAIS functional entities

The current SIP definitions of existing SCAPE repositories have significant differences: In RODA for example a SIP is a compressed ZIP file that contains a METS envelope, in Rosetta a SIP may contain several IEs (Intellectual Entities). An AIP contains technical metadata and metadata important for long term archiving.

2.2 METS

The METS specification (in XML format) conveys the metadata to manage digital objects within a repository and to exchange such objects between repositories and / or the users. The METS[1] standard was designed to comply with the OAIS 'Information Package' definition.

2.3 A Mets Document

A typical METS document consists of up to 6 sections. All sections are optional except the Structural Map section, which is mandatory.

	Description	Format
Header	Dates (update, creation), status, author, role	
Descriptive Metadata	No vocabulary or syntax for encoding descriptive metadata	Any form, Dublin Core, MARC, MODS, EAD etc.
Administrative Metadata	No vocabulary or syntax for encoding administrative metadata.	
- **Technical metadata**	ditto	Any form, e.g. MIX, FITS, PREMIS Object Metadata …
- **Source metadata**	ditto info (descriptive, rights, technical) about the analogue source used to generate the	Any form, e.g. PREMIS

[1] Metadata Encoding and Transmission Standard

	digital object	
- **Rights metadata**	ditto	Any form, e.g. indices, copyrightMD, PREMIS Rights Metadata
- **Digital provenance metadata**	ditto	Any form, e.g. PREMIS Event Metadata
File Section	All files that comprise the content of the digital entity. Files are ordered in groups (tiff, jpeg etc.) File element may refer to an external file.	
Structural Map section	Specifies the structure of the digital entity. Specifies how the files fit into this structure. More than one structure possible, e.g. logical and physical.	
Behaviour Section	List all dissemination behaviours	

2.4 PREMIS

PREMIS is an acronym for **Pre**servation **M**etadata: **I**mplementation **S**trategies. The Data Dictionary of PREMIS defines preservation metadata and provides an XML schema[2]. The PREMIS Data Model[3] defines Intellectual Entities, Objects, Rights, Agents and Events.

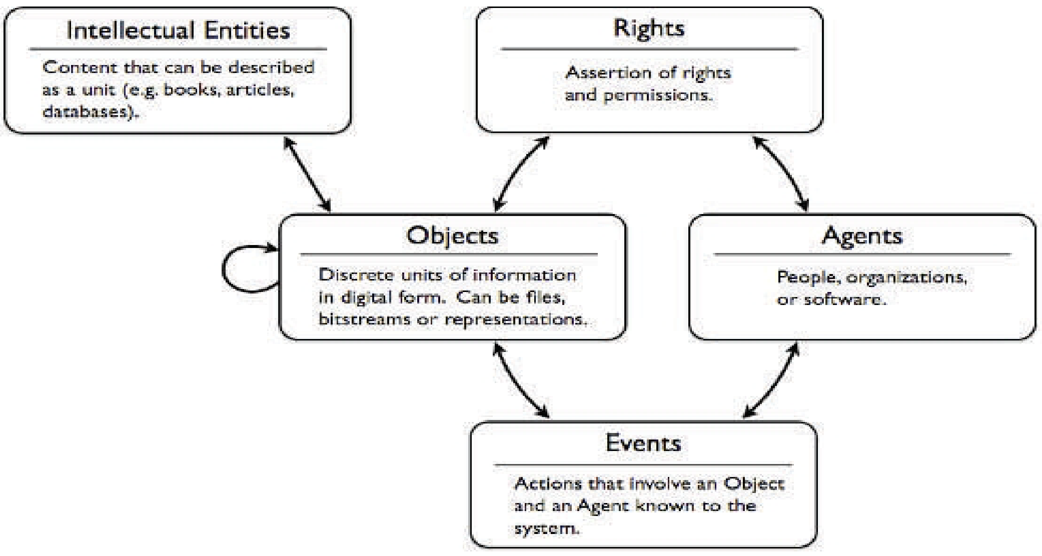

Intellectual Entities
Content that can be described as a unit (e.g. books, articles, databases).
Rights
Assertion of rights and permissions.
Objects
Discrete units of information in digital form. Can be files, bitstreams or representations.
Agents
People, organizations, or software.
Events
Actions that involve an Object and an Agent known to the system.

Figure 2-2: The PREMIS Data Model

[2] http://www.loc.gov/standards/premis/premis.xsd
[3] http://www.loc.gov/standards/premis/

An Object in PREMIS has three subtypes: file, representation and bitstream.

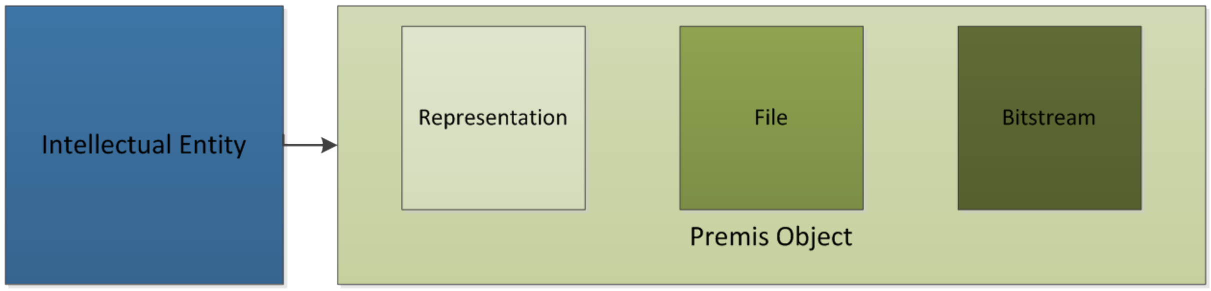


Figure 2-3: The PREMIS Data Model with Intellectual Entity and the Object with its subtypes: Representation, File and Bitstream.

The Intellectual Entity is, e.g., a book, a map, photograph or database, etc. The Representation of an IE is a set of Files including structural Metadata (e.g. described by METS). A File is a named and ordered sequence of bytes known by the operating system that can be written, read and copied. Bit streams represent data within a file, e.g., a jpeg in a PDF document, audio data within a WAVE file or graphics within a Word document.

2.5 METS & PREMIS

METS as an XML container for structuring metadata in different formats is often used in conjunction with the PREMIS standard for preservation metadata. But one has to consider the existing overlap of the METS and PREMIS definitions. There are a few documents available describing best practices and guidelines of how to use PREMIS within METS, see for example[4].

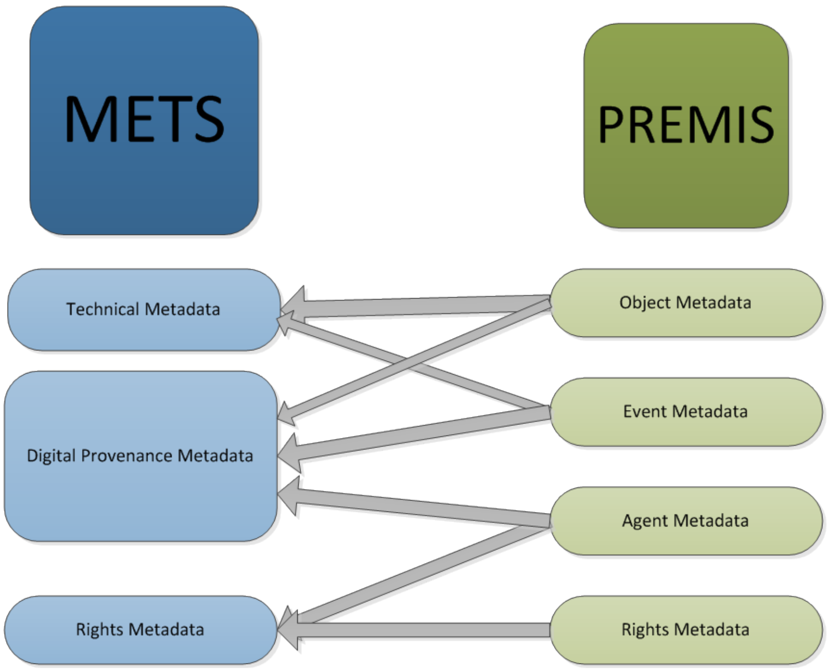


Figure 2-4: Mapping PREMIS entities to METS metadata sections. Thick arrows show applicable subsection in METS for the named PREMIS entities; the thin arrow shows links from one PREMIS entity to another METS subsection.

[4] http://www.loc.gov/standards/premis/guidelines-premismets.pdf

The following 13 checkpoints may help when dealing with METS and PREMIS:

1. How does the profile relate to other METS profiles
2. What schemas (PREMIS, MOS, MIX) are used and where are they located
3. What controlled vocabularies for PREMIS semantic units are used and where are they located?
4. Is PREMIS information wrapped into or referenced from the METS document?
5. Is PREMIS information bundled or distributed in several places in the METS document?
6. IS PREMIS information placed in separate amdSec elements or amdSec sub-elements?
7. Is technical metadata recorded in separate techMD sections or with PREMIS objectCharacteristicExtension?
8. What PREMIS semantic units does the profile require or recommend?
9. Are relationships between objects expressed using METS div elements, PREMIS relationships, or both?
10. What level of object does PREMIS information describe?
11. How are PREMIS linking identifiers, IDREFs, and PREMIS identifiers used?
12. How are PREMIS-METS redundancies handled?
13. What metadata tools or applications are used?

For a detailed discussion of these checkpoints (with examples) please refer to "A Checklist for Documenting PREMIS - METS Decisions in a METS Profile".[5]

3 Digital Object Model of SCAPE Repositories

The SCAPE partners are using different repository implementation such as Rosetta (ExLibris), RODA (Keeps) and eSciDoc (FIZ Karlsruhe). All of these repositories do have their own Digital Object Model. We will briefly describe these models in the following section.

3.2 Rosetta

Ex Libris Rosetta is a digital-object preservation solution that conforms to the ISO-recognized Open Archival Information System (OAIS). Rosetta allows the SIP and the DIP to have a variety of formats and structures and provides an SDK to support this. We describe below the AIP, which is stored in a METS XML file in Rosetta's Permanent Repository module. Each AIP describes one IE (intellectual entity).

The METS XML is generated in the Staging module during the SIP processing. During processing, the IE information is kept and managed in the database. By the time the SIP is moved to the permanent repository, the METS XML contains all the information regarding the IE, collected from the different database tables.

The information on the METS XML can be reloaded back into the database when the IE is brought from the permanent repository for maintenance (preservation actions, adding representations, and so forth).

The following diagram shows the flow between the three types of information packages:

[5] http://www.loc.gov/standards/premis/premis_mets_checklist.pdf

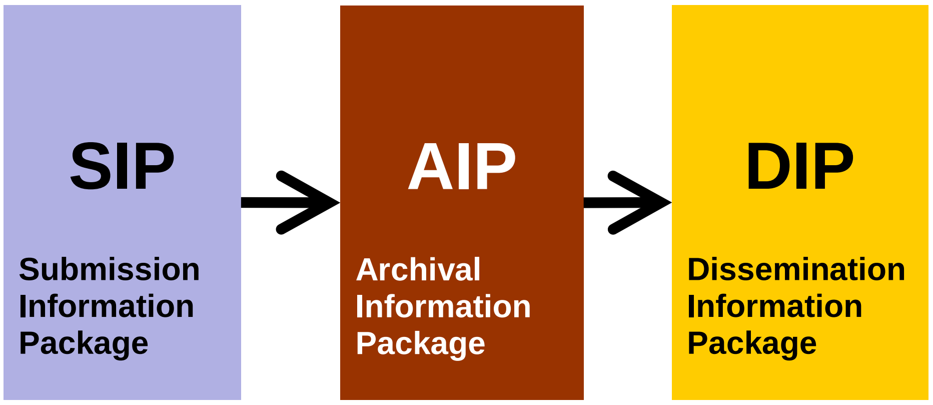

SIP
Submission Information Package
AIP
Archival Information Package
DIP
Dissemination Information Package

Figure 3-1: Flow between the three types of information packages

The AIP is stored in a METS XML file and can be viewed in the Permanent Repository module of Rosetta. Each AIP relates to one IE (intellectual entity).

Structure

Rosetta AIP data model that is based on the PREMIS reference model covers four levels of objects. [6] The following figure illustrates the four entities of the AIP data model:

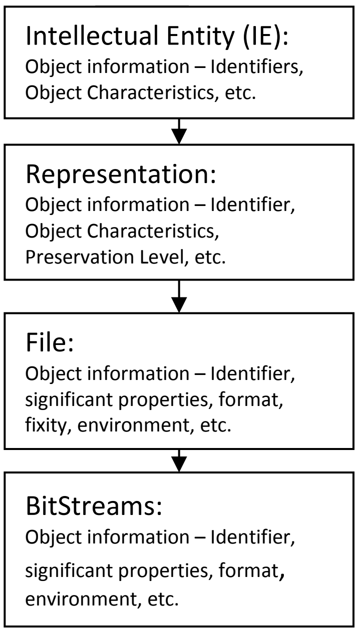

Intellectual Entity (IE):
Object information – Identifiers, Object Characteristics, etc.
Representation:
Object information – Identifier, Object Characteristics, Preservation Level, etc.
File:
Object information – Identifier, significant properties, format, fixity, environment, etc.
BitStreams:
Object information – Identifier, significant properties, format, environment, etc.

Figure 3-2: Four entities of the AIP data model

[6] Further information related to the PREMIS reference model can be found at: http://www.loc.gov/standards/premis/.

The entities in the data model are defined as follows:

- **Intellectual Entity** – A set of content that is considered a single intellectual unit for purposes of management and description – for example, a particular book, map, photograph, or database. An intellectual entity may have one or more digital representations.
- **Representations** – A Representation is the set of files, including structural metadata, needed for a complete and reasonable rendition of an intellectual entity. There can be more than one representation for the same intellectual entity. For example, a journal article may be complete in one PDF file and this single file will then constitute the representation. However, another journal article may consist of one SGML file and two image files. In this case, these three files will constitute the representation. A third article may be represented by one TIFF image for each of 12 pages plus an XML file of structural metadata showing the order of the pages. In this case, 13 files will constitute the representation. (PREMIS data dictionary, p. 14)
- **Files** – A file is a named and ordered sequence of bytes that is known by an operating system. A file can be zero or more bytes and has a file format, access permissions, and file system characteristics such as size and last modification date.
- **Bitstreams** – A bitstream is contiguous or non-contiguous data within a file that has meaningful common properties for preservation purposes. A bitstream cannot be transformed into a standalone file without the addition of file structure (headers, and so forth) and/or reformatting to comply with a particular file format.

  A bitstream is defined in the PREMIS data model as a set of bits embedded within a file. This differs from common usage, where a bitstream could, in theory, span more than one file.

  A good example of a file with embedded bitstreams is a TIFF file containing two images.

  According to the TIFF file format specification, a TIFF file must contain a header that includes information about the file. It may then contain one or more images. In the data model, each of these images is a bitstream and can have properties such as identifiers, location, inhibitors, and detailed technical metadata (for example, colour space).

  Some bitstreams have the same properties as files and some do not. The image embedded within the TIFF file clearly has properties that are different from the file itself. However, three TIFF files can also be aggregated within a larger TAR file. In this case, the three TIFF files are filestreams, but they have all the properties of TIFF files. [7]

  Rosetta bitstream functionality is limited to filestream only. Real bitstreams (embedded objects within a file) are functionally not supported however from a Data Model perspective; the Data Model serves both types of bitstreams

Rosetta uses METS as a container for the IE as an AIP. Further information related to the METS reference model can be found at [8]

The METS schema contains three types of metadata: Descriptive, Administrative, and Structural Map.

[7] http://www.loc.gov/standards/premis/v2/premis-2-1.pdf

[8] http://www.loc.gov/standards/mets/

The following table illustrates which metadata type is applicable to each object type:

	Descriptive	Administrative	Structural Map
IE	√	√	-
Representation	-	√	√
File	-	√	-
Bitstream	-	√	-

The following graphic illustrates the METS Container in Rosetta:

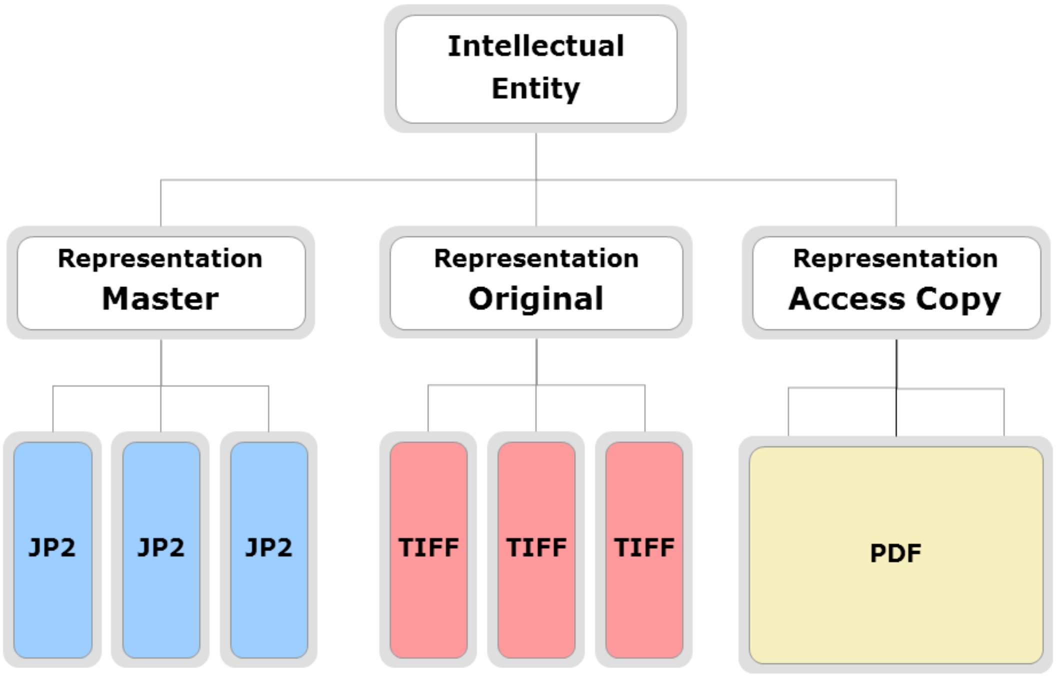


Figure 3-3: An Illustration of a Mets Container in Rosetta

3.3 RODA

RODA is an open source digital repository with long-term preservation and authenticity as its primary objectives. Created by the Portuguese National Archives in partnership with the University of Minho, it was designed to support the most recent archival standards and become a trustworthy digital repository.

RODA has been developed to be a complete digital repository providing functionality for all the main units that compose the OAIS reference model. As an OAIS compliant repository, RODA defines a Submission Information Package (SIP), an Archival Information Package (AIP) and a Dissemination Information Package (DIP) format for ingest, archival and dissemination of information, respectively.
In the following sections RODA SIP and AIP are further described.

SIP

New data is submitted to the repository in the shape of Submission Information Packages (SIP). When the ingest process terminates, SIPs are transformed into Archival Information Packages (AIP), i.e., the actual packages that will be kept in the repository. Associated with the AIP is the structural, technical and preservation metadata, as they are essential for carrying out preservation activities.

The SIP is composed of one or more digital representations and all of the associated metadata, packaged inside a METS envelope. Producers take advantage of a small application called RODA-in that allows them to create these packages. The structure of a SIP supported by RODA is depicted in Figure 3-4. The RODA SIP is basically a compressed ZIP file containing a METS envelope, the set of files that compose the representations and a series of metadata records. Within the SIP there should be at least one descriptive metadata record in EAD-Component[9] format.

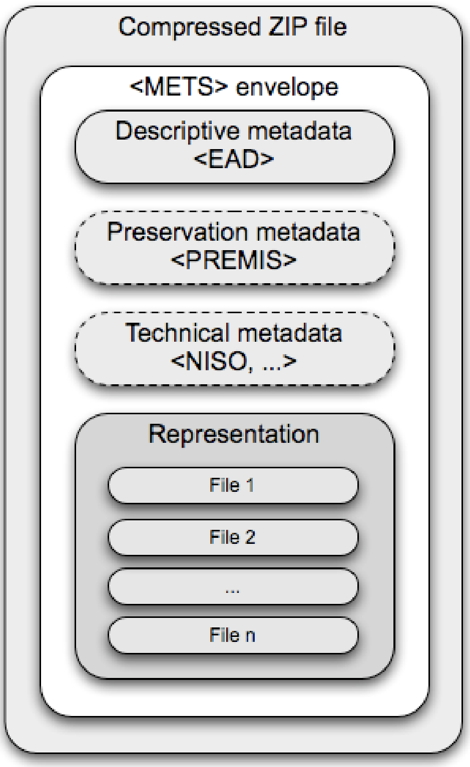

Compressed ZIP file
<METS> envelope
Descriptive metadata <EAD>
Preservation metadata <PREMIS>
Technical metadata <NISO, ...>
Representation
File 1
File 2
...
File n

Figure 3-4: Structure of a Submission Information Package in RODA using a METS envelope

One may also find preservation and technical metadata inside a submission package, although this last set of metadata is not mandatory as is seldom created by producers. Nevertheless, RODA supports those additional SIP elements for special situations such as repository succession, e.g., when ingested items belong to a repository that is to be deactivated.

The main file inside a SIP is the METS envelope file (METS.xml) that structures the SIP information. This envelope file references the descriptive metadata that can reference one or more representations that in turn reference the files. Representations can also reference preservation and technical metadata.

[9] An EAD record does not describe a single representation. In fact, EAD is used to describe an entire collection of representations. In the SIP is included only a segment of EAD that is sufficient to describe one representation, i.e. a <c> element and all its sub-elements. The team has called this subset of the EAD an EAD-Component.

The following snippet is an example of a METS envelope file representing a SIP in RODA:

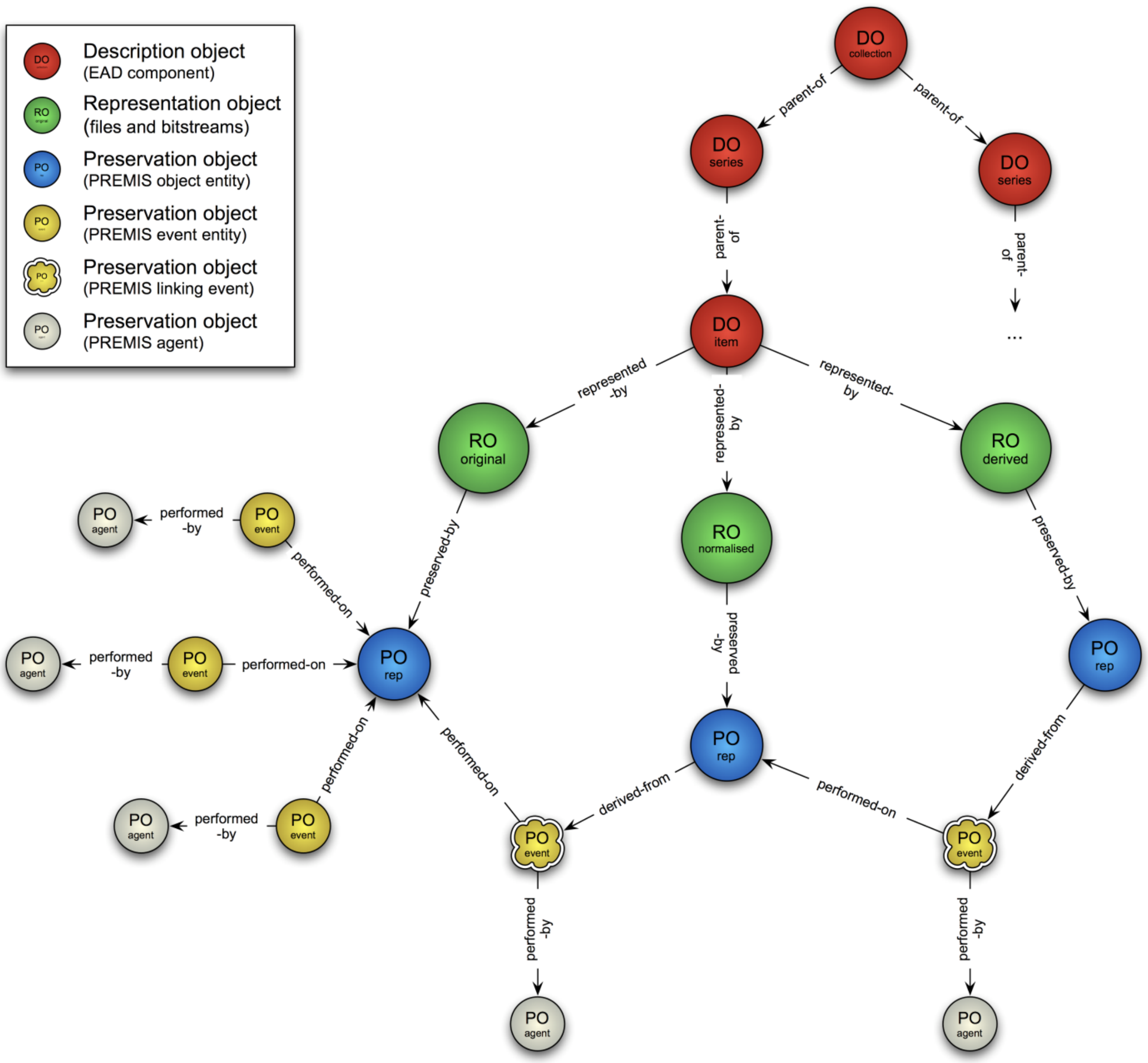
```xml
<?xml version="1.0" encoding="UTF-8"?>
<mets PROFILE="RODA_SIP" xmlns="http://www.loc.gov/METS/"
xmlns:xlin="http://www.w3.org/1999/xlink">
  <metsHdr>
    <agent ROLE="CREATOR">
      <name>RODA Common SIP Utility</name>
    </agent>
  </metsHdr>
  <dmdSec ID="EADC-WALLPAPER--1664823803">
    <mdRef LOCTYPE="URL" MDTYPE="OTHER" xlin:href="eadc5700174555992435226.tmp"
      LABEL="roda:d" CHECKSUMTYPE="MD5"
      CHECKSUM="12069DF39F92FF1682FACE1E1FC2B712"/>
  </dmdSec>
  <dmdSec ID="EADC-NASA-1691196776" GROUPID="EADC-WALLPAPER--1664823803">
    <mdRef LOCTYPE="URL" MDTYPE="OTHER" xlin:href="eadc2713347945811457232.tmp"
      LABEL="roda:d" CHECKSUMTYPE="MD5"
      CHECKSUM="EEC03168F8C3D30D03A9089F1DF31685"/>
  </dmdSec>
  <fileSec>
    <fileGrp>
      <file ID="R2012-03-28T14.44.44.49Z-F0" MIMETYPE="application/octet-stream"
             CHECKSUMTYPE="MD5" CHECKSUM="5F4CDB67BC8EA454A70FA448BDF71B31">
        <FLocat LOCTYPE="URL" xlin:href="R2012-03-28T14.44.44.49Z/F0"
             xlin:title="METS.xml"/>
      </file>
      <file ID="R2012-03-28T14.44.44.49Z-F1" MIMETYPE="image/png"
             CHECKSUMTYPE="MD5" CHECKSUM="EEF92204A161CC8C157B4D3F8B7FBA74">
        <FLocat LOCTYPE="URL" xlin:href="R2012-03-28T14.44.44.49Z/F1"
          xlin:title="magField_3d_02Left_lrg.png"/>
      </file>
      <file ID="R2012-03-28T14.44.44.49Z-F3" MIMETYPE="image/jpeg"
             CHECKSUMTYPE="MD5" CHECKSUM="829AD7D9A296FFFE86B4E01E8E6D9178">
        <FLocat LOCTYPE="URL" xlin:href="R2012-03-28T14.44.44.49Z/F3"
          xlin:title="keithburns-1920.jpg"/>
      </file>
    </fileGrp>
  </fileSec>
  <structMap>
    <div ID="Representations">
      <div ID="R2012-03-28T14.44.44.49Z"
             TYPE="roda:r:digitalized_work:image/mets+misc"
             DMDID="EADC-NASA-1691196776" xlin:label="original">
        <fptr FILEID="R2012-03-28T14.44.44.49Z-F0"/>
        <fptr FILEID="R2012-03-28T14.44.44.49Z-F1"/>
        <fptr FILEID="R2012-03-28T14.44.44.49Z-F3"/>
      </div>
    </div>
  </structMap>
</mets>
```

After a successful validation, the SIP is then taken apart and each of its constituents is integrated into the repository. After this procedure, the SIP becomes an AIP and is ready to be disseminated to potential consumers that have clearance to access that information.

AIP

Unlike the SIP, RODA's AIP does not rely on METS to structure the information. Instead, it uses the Fedora Commons RDF linking mechanism to keep the reference integrity between the several AIP constituents.

RODA’s content model is atomistic and very much PREMIS-oriented (Figure 3-5). Each intellectual entity has an EAD-component metadata record (DO nodes in Figure 3-5) describing it. These records are organized hierarchically in order to constitute a full archival description, but are kept separately within the Fedora Commons content model.



Description object (EAD component)
Representation object (files and bitstreams)
Preservation object (PREMIS object entity)
Preservation object (PREMIS event entity)
Preservation object (PREMIS linking event)
Preservation object (PREMIS agent)
DO collection
parent-of
DO series
parent-of
DO series
parent-of
...
DO item
represented-by
RO original
RO normalised
RO derived
preserved-by
PO rep
performed-on
PO event
performed-by
PO agent
derived-from

Figure 3-5: RODA’s molecular content model featuring EAD-components, digital

Additionally, each leaf record (i.e., a file or an item) is linked to a representation object (RO nodes in the figure), i.e., a Fedora object that embeds all the files and bit-streams that compose the digital

representation. Finally, each of these objects is linked together by a set of PREMIS entities that maintain information about the digital object's provenance and history of events (PO nodes).
Each preservation event that takes place inside the repository is recorded as a new preservation-event node (i.e., PO event nodes in the figure). Special events, like format migrations, establish relationships between two preservation-representation nodes. These are called linking events. A suitable agent, either a system user or an automatically triggered software application, executes each preservation event. The agent that triggered the event is recorded in PO agent nodes.

3.4 eSciDoc

eSciDoc is a joint project of the Max Planck Society and FIZ Karlsruhe. It primarily focuses on the support of the scientific life cycle but can be used for other approaches as well. It comprises core functionality including a Fedora Commons repository, a set of complementing services, and an application build on top of the infrastructure and the services that enable innovative eScience scenarios, such as support of scientific collaboration and interdisciplinary research. Scientists, librarians, and software developers can work with research data, create novel forms of publications, and establish new ways of scientific and scholarly communication.[10]

Even though eSciDoc covers many disciplines and use cases in the field of eScience, eSciDoc has only three generic object patterns:

- Context (administrative container)
- Container (aggregations, e.g., collections, bundles)
- Item

Each content resource (Item or Container) is maintained in a single administrative Context. The two content resources, Item and Container are defined as follows:

- An *Item* resource consists of metadata records (e.g., eSciDoc publication metadata, MODS record, Dublin Core record) and optionally of *Components* that represent the actual content (e.g. PDF file, JPEG file, XML file).
- A *Container* resource is an aggregation of other resources that allows for aggregating other items or containers. Like the Item resource, a Container may have multiple metadata records that describe it.

Items and Containers are very generic resources and they do not speak for themselves about the content they represent or about their own structure, e.g., what kind of metadata may be associated with them, what kind of members a container aggregates, or what kind of resources they represent semantically. Therefore, eSciDoc logical data model introduced the concept of a Content model. Each Item or a Container has to claim that it is an instance of exactly one content model. There is no limitation on the number of instances that may claim to be of a specific content model.

Content model defines in general:

- The type and structure of the content resources (Item, Container, Components), and
- A set of services that may be associated with the content resources.

[10] https://www.escidoc.org/

The following graphic illustrates the eSciDoc object patterns and their relationships for a specific example with a Context “My Science Lab”:

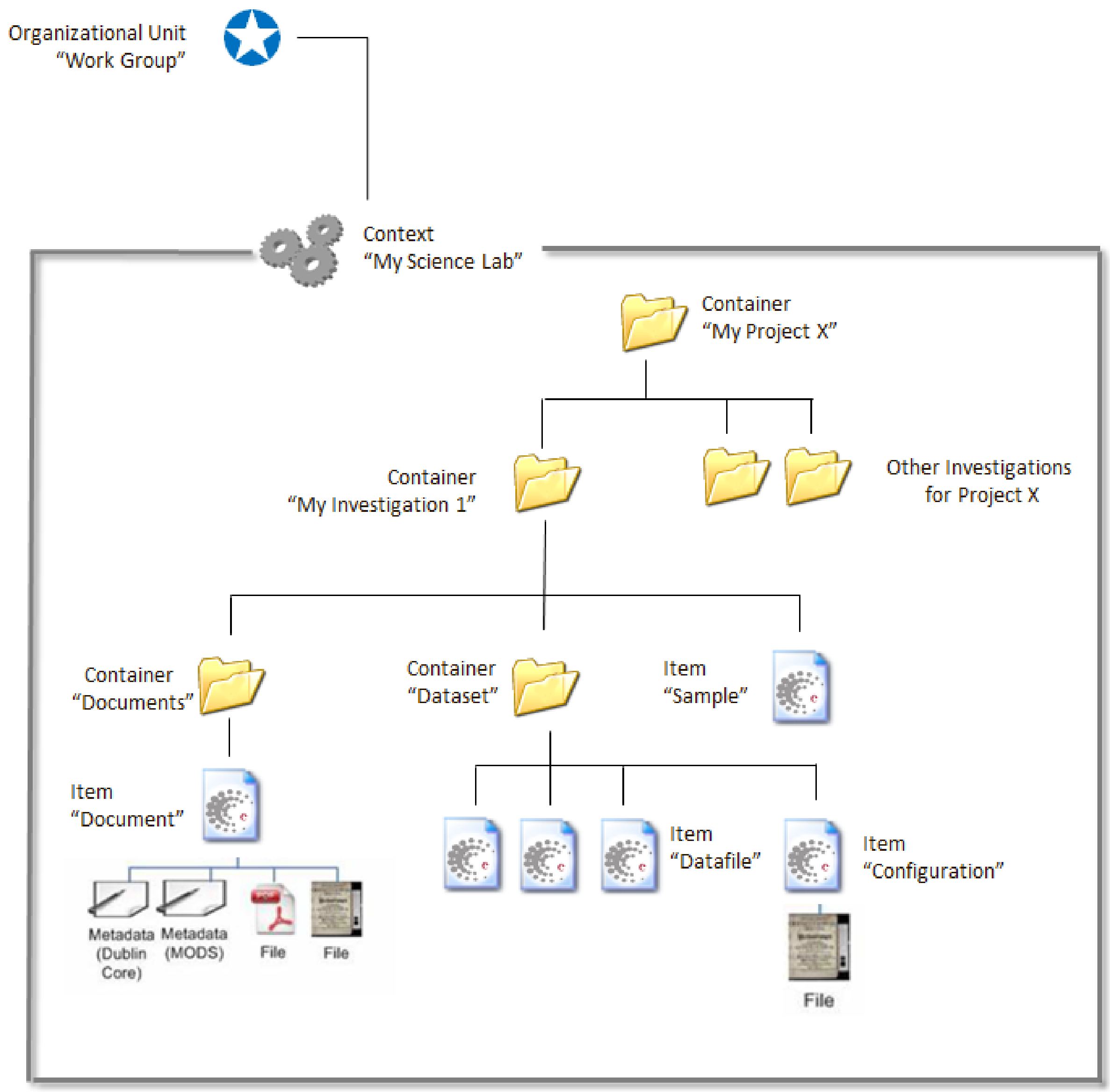


Figure 3-6: sample view on the generic object patterns and their relationship to each other of eSciDoc for a Context “My Science Lab”.

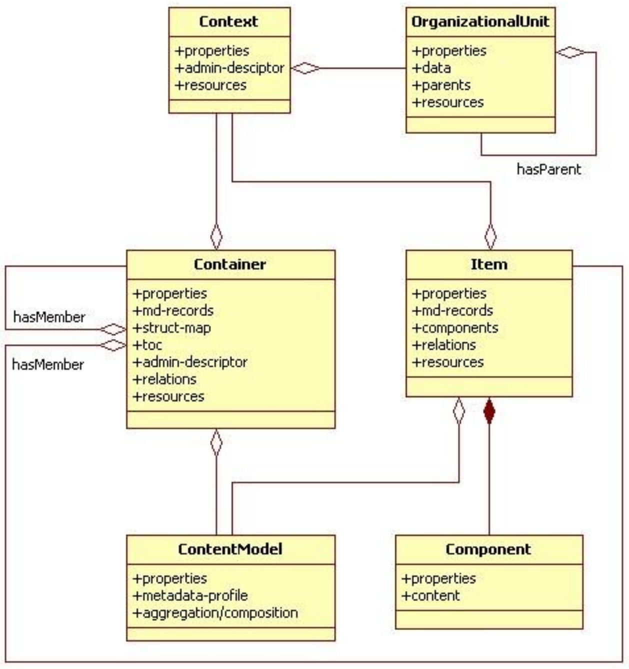


Figure 3-7: A technical view of the eSciDoc Data Model

eSciDoc needs to map the METS profile to the existing eSciDoc data model. One eSciDoc item may represent an Intellectual Entity and the eSciDoc components may be mapped to the representations. It is also possible to have an eSciDoc Container represent an Intellectual Entity and have the eSciDoc Items map to the representations and have its eSciDoc Components map to files.

A possible mapping of the eSciDoc Data Model to PREMIS terms:

eSciDoc	PREMIS
Container	Intellectual Entity
Item	Representation / Intellectual Entity
Component	Representation / File / Bitstream

4 The SCAPE Digital Object Model

In this model each Intellectual Entity will be represented by one METS file, and each Representation File and Bitstream will be described by technical metadata.

4.1 Requirements METS

In SCAPE we define one METS profile for all use cases and OAIS Information Packages.
The different optional and mandatory properties of the METS profile for SIP and AIP will be indicated in the profile itself. For SIP a minimal set of elements are mandatory and for the AIP we will have all elements that are needed for preservation are mandatory.

Formats for all relevant Metadata in the SCAPE METS profile

- Descriptive Metadata (Dublin Core, EAD, MODS)
- Technical Metadata (PREMIS, MIX, VideoMD, AudioMD, textMD, JHOVE)
- Digital Provenance Metadata (PREMIS event and agents)
- Source Metadata (contains descriptive, rights or technical metadata)
- Rights Metadata (PREMIS rights)

In order to map PREMIS entities to METS we follow the guidelines described in “Guidelines for using PREMIS with METS for exchange”[11]. The following table illustrates the mapping:

PREMIS	METS
premis:object	techMD or digiProvMD
premis:event	digiProvMD
premis:rights	rightsMD
premis:agent	digiProvMD or rightsMD

A generic METS profile that uses PREMIS is available[12].

The following METS XML snippet shows the basic structure of a METS XML document for one Intellectual Entity “fiz.karlsruhe.09601” with two objects. Note: the sections are empty.

[11] http://www.loc.gov/standards/premis/guidelines-premismets.pdf
[12] http://www.loc.gov/standards/premis/louis-2-0.xml

```
<mets:mets PROFILE="SCAPE" OBJID="fiz.karlsruhe.09601"
xsi:schemaLocation="http://www.loc.gov/METS/
http://www.loc.gov/standards/mets/mets.xsd">
<mets:dmdSec ID="DC"></mets:dmdSec>
<mets:amdSec>
	<mets:techMD ID="object1"></mets:techMD>
	<mets:rightsMD ID="rights1"></mets:rightsMD>
	<mets:sourceMD ID="source1"></mets:sourceMD>
	<mets:digiprovMD ID="event1"></mets:digiprovMD>
	<mets:techMD ID="object2"></mets:techMD>
	<mets:rightsMD ID="rights2"></mets:rightsMD>
	<mets:sourceMD ID="source2"></mets:sourceMD>
	<mets:digiprovMD ID="event2"></mets:digiprovMD>
</mets:amdSec>
<mets:fileSec></mets:fileSec>
<mets:structMap></mets:structMap>
</mets:mets>
```

The following example shows the usage of PREMIS within the METS techMD section:

```
<mets:techMD ID="object1">
<mets:mdWrap MDTYPE="PREMIS:OBJECT">
<mets:xmlData>
<premis:object xsi:type="premis:file"
xsi:schemaLocation="info:lc/xmlns/premis-v2
http://www.loc.gov/standards/premis/v2/premis-v2-0.xsd">
	<premis:objectIdentifier></premis:objectIdentifier>
	<premis:preservationLevel></premis:preservationLevel>
	<premis:significantProperties></premis:significantProperties>
	<premis:objectCharacteristics></premis:objectCharacteristics>
	<premis:originalName></premis:originalName>
	<premis:storage></premis:storage>
	<premis:environment></premis:environment>
	<premis:relationship></premis:relationship>
	<premis:linkingEventIdentifier></premis:linkingEventIdentifier>
	<premis:linkingIntellectualEntityIdentifier></premis:linkingInte
llectualEntityIdentifier>
</premis:object>
</mets:xmlData>
</mets:mdWrap>
</mets:techMD>
```

4.2 Example METS Profile

In order to make Platform implementations relatively easy to interchange, particularly between different tools, a METS profile for SCAPE has been defined with the following characteristics:

A METS file adhering to this profile

- ***may*** contain a “<metsHdr>” element,
- ***must*** contain one and only one “<dmdSec>” element,
- ***must*** use the “<mdWrap>” or <mdRef> element for metadata,
- ***must*** have one “<amdSec>” element [not for SIP],
- ***must*** have one <techMD> for each file [not for SIP],
- ***may*** contain <rightsMD>, <sourceMD>, <digiprovMD> for each file,
- ***must*** have a “<fileGrp>” element for each Representation,
- ***must*** have a “<file>” element containing an “<FLocat>” element with location information as a URL in the “xlink:href” attribute,
- ***must*** have a “<structMap>” element,
- ***must*** not have a “<structLink>” element,
- ***must*** not have a “<behaviourSec>” element,
- ***must*** wrap descriptive, technical, rights, source and provenance metadata in a “<mdWrap>” or <mdRef> element composed of PREMIS elements,
- ***must*** only use descriptive metadata composed of Dublin Core terms,
- ***must*** only use technical metadata composed of NISO Z39.87 terms for digital still images (MIX), TextMD for textual representations [...],
- ***must*** only use provenance metadata composed of PREMIS events,
- ***must*** only use source metadata composed of Dublin Core terms, and
- ***must*** only use rights metadata composed of PREMIS rights.

The following figure shows the SCAPE METS profile with optional (grey), mandatory (blue) and reference implementation (green) elements.

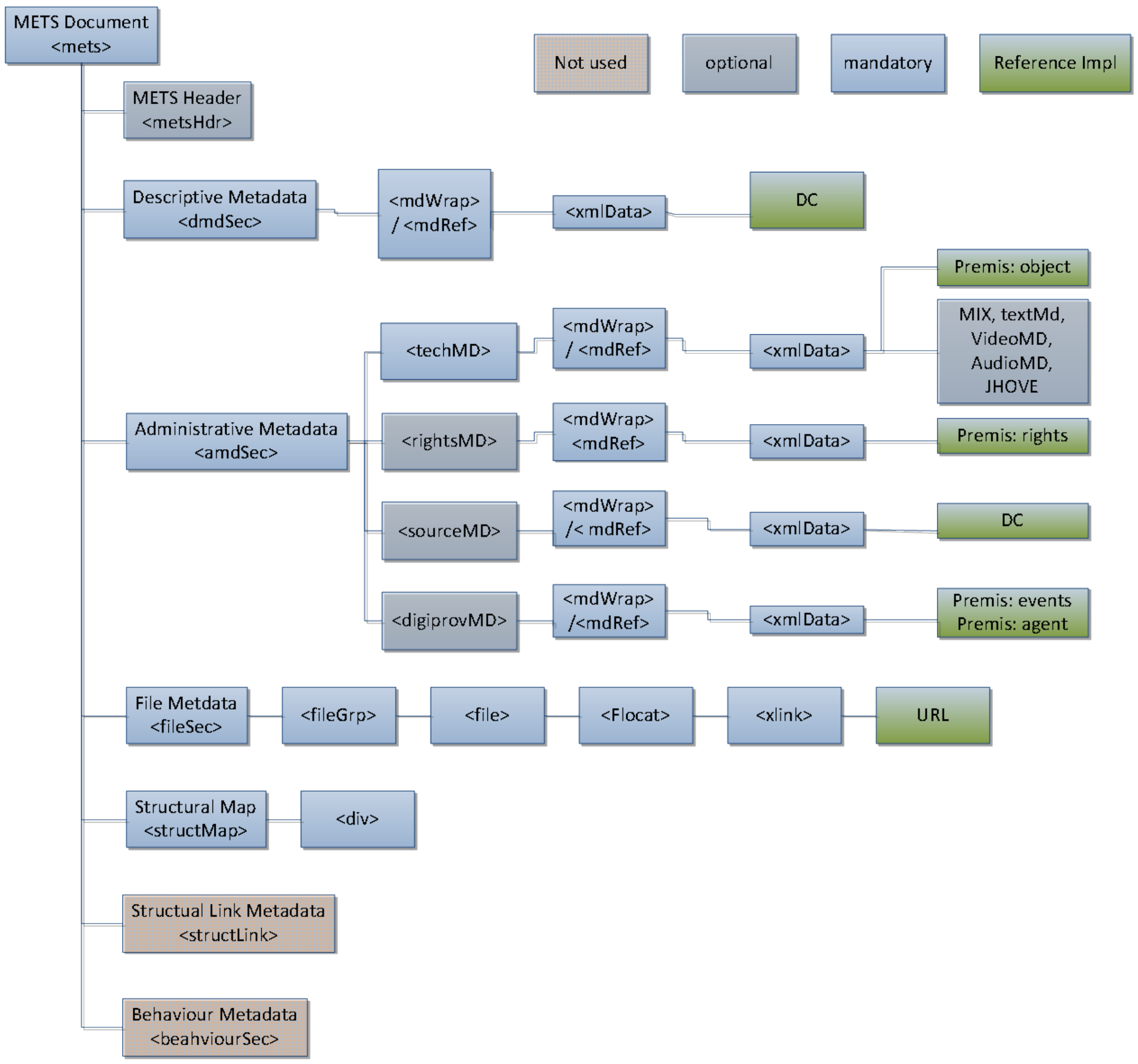


Figure 4-1: SCAPE METS Profile

4.3 METS StructMap

The METS document incudes exactly one <structmap> element, which is used to map the structure of the intellectual entity into a flat METS XML representation. The structure map consists of nested <div> elements that correspond to the structure of an Intellectual Entity, i.e., Intellectual Entity, Representation, File, Bitstream. There is exactly one <div> element on the top level corresponding to the Intellectual entity. This <div> contains nested <div> elements for each representation; these are linked via identifiers to the metadata describing the representation. Each representation contains a set of nested <div> elements for each File, linking to its technical metadata. Each of the <div> elements on the file level contains exactly one <fptr> element for the identification of the File. The <div>-element on the file level in turn can contain nested <div> elements for each Bitstream in the File.

The following snippet shows an example of the METS StructMap:

```
<div id="1" type="IntellectualEntity" label="entity_1" dmdid="dmd-1">
    <div id="2" type="Representation" label="rep-1">
        <div id="3" type="techMD" admid="tech-1"/>
        <div id="4" type="rightsMD" admid="reghts-1"/>
        <div id="5" type="digiProvMD" admid="digiprov-1"/>
        <div id="6" type="sourceMD" admid="source-1" order="0"/>
        <div id="7" type="File">
             <fptr fileid="file-1"/>
             <div id="8" type="techMD" admid="tech-2"/>
               <div id="9" type="Bitstream" >
                 <div id="10" type="techMD" admid="tech-3"/>
               </div>
        </div>
    </div>
</div>
```

4.4 METS and PREMIS Identifiers

Each METS document must be assigned a persistent and unique identifier. This identifier must be locally and globally resolvable. Possible identifier schemes are: OCLC Purls[13], CNRI Handles[14], DOI[15] etc. But it is also possible to use just UUID[16] to assign a unique - but not global - identifier. A further requirement is that the identifier must be a string. Further Information can be found at [17] .

For a SIP the METS identifier is optional assuming that the repository will assign an identifier on submission. The identifier will be recorded in the OBJID attribute of each METS document.

The PREMIS identifiers such as objectIdentifier, agentIdentifier, permissionStatementIdentifier must be consistent within each METS document, but don't have to be globally or locally resolvable.

4.5 Extension Schemas

- DC – Dublin core
    - May be included in the dmdMD element. Each METS document must contain a dmdSec that describes the entire package.
    - http://dublincore.org/schemas/xmls/

- PREMIS
    - See table in section 8.1 for the usage of PREMIS in METS.
    - http://www.loc.gov/standards/premis/v2/premis-v2-1.xsd

- Technical Metadata for Digital Still Images (MIX) - NISO Data Dictionary
    - May be included in the techMD element

[13] http://purl.org/docs/index.html
[14] http://www.handle.net/
[15] http://www.doi.org/
[16] http://www.itu.int/ITU-T/studygroups/com17/oid.html
[17] http://en.wikipedia.org/wiki/Universally_unique_identifier

- http://www.loc.gov/standars/mix/mix.xsd

- textMD – Text Metadata Schema
    - May be included in the techMD element
    - http://dlib.nyu.edu/METS/textmd.xsd

- VideoMD – Video Technical Metadata Extension Schema
    - May be included in the techMD element
    - http://lcweb2.loc.gov/mets/Schemas/VMD.xsd

- AudioMD – Audo Technical Metadata Extension Schema
    - May be included in the techMD element
    - http://lcweb2.loc.gov/mets/Schemas/AMD.xsd

- JHOVE XML handler output schema
    - May be included in the techMD element
    - http://hul.harvard.edu/ois/xml/xsd/jhove/jhove.xsd
- FITS File Information Tool Set
    - May be used in the techMD element
    - http://hul.harvard.edu/ois/xml/xsd/fits/fits_output.xsd

4.6 Requirements for the OAIS Information Packages

We need to define the three Information Packages described by OAIS for SCAPE

4.7 Definition of a SIP

The simplest SIP contains a header and content (files). A realistic example for a SIP is illustrated in the RODA section of this document. This definition of a SIP can be used for the SCAPE Loader Application as well. The following figure shows the minimum definition of a SIP METS profile used within SCAPE:

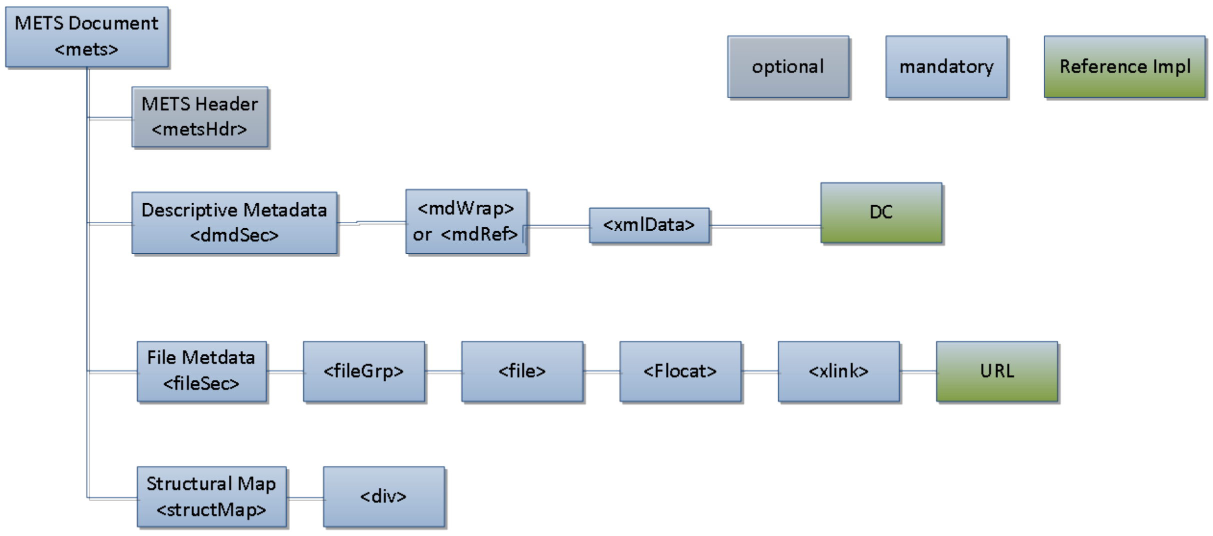


Figure 4-2: SCAPE METS profile of a SIP

4.8 Definition of a AIP

An AIP is an archived Intellectual Entity and should therefore contain technical metadata and digital preservation metadata. An AIP may also contain information about the preservation plan associated with the Intellectual Entity. Please refer to section 4.2 for further information about the METS profile of an AIP.

4.9 Definition of a DIP

A Dissemination Information Package (DIP) in SCAPE will contain the same information as an Archival Information Package (AIP).

4.10 Preservation Plans

Preservation Plans will not be described by this SCAPE METS profile but do have their own XML schema definition[18], although it can be wrapped in a METS container. If a Plan gets executed the provenance information and the information about the plan executed gets stored in the digital provenance section in the AIP of the digital object.

4.11 Summary

The following graphic illustrates the OAIS Information packages and the relationship with an Intellectual Entity:

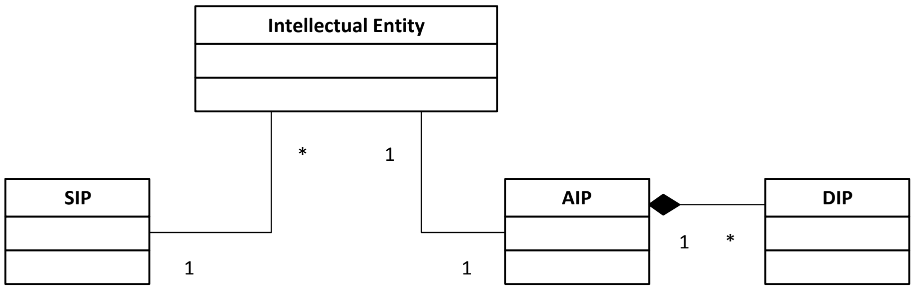


Figure 4-3: The possible relationship of an Intellectual Entity with a SIP, AIP and DIP

One SIP can hold several Intellectual Entities. One Intellectual Entity is represented by one AIP. A DIP is either a one-to-one representation of an AIP or can be just part of an AIP.

The following table summarizes the definition of a METS profile of a SIP and AIP:

METS section	AIP	SIP
<metsHdr>	optional	optional
<dmdSec>	mandatory	mandatory
<amdSec>	mandatory	-
<fileSec>	mandatory	mandatory
<structMap>	mandatory	mandatory
<structLink>	-	-
<behaviourSec>	-	-

Figure 4-4: Repository System showing SCAPE SIP and DIP

[18] http://www.ifs.tuwien.ac.at/dp/plato/schemas/plato-3.0.1.xsd - this definition will change in the future.

5 Implementation of the SCAPE Digital Object Model

The SCAPE Digital Object Model has been implemented in JAVA and is published on Github.[19]
This implementation helps the developer to serialize and deserialize intellectual entities in and from METS representations.
The following snippet shows the serialization of an intellectual entity:

```
IntellectualEntity entity = TestUtil.createRandomEntity();
SCAPEMarshaller.getInstance().serialize(entity, System.out);
```

An example for deserialization looks like this:

```
InputStream in = new FileInputStream("entity-1.xml");
IntellectualEntity deserialized = SCAPEMarshaller.getInstance()
        .deserialize(IntellectualEntity.class, in);
```

This project contains also test classes to create Mets representations of intellectual entities to test the implementations of the Data Connector API. One method is provided here as an example:

```
public static IntellectualEntity createTestEntity(String entityId)
            throws JAXBException {
        BitStream bs_1 =
                new BitStream.Builder().identifier(
                        new Identifier("bitstream-1")).technical(
                        TestUtil.createFITSRecord()).build();

        File f =
                new File.Builder().mimetype("image/png").bitStreams(
                        Arrays.asList(bs_1)).identifier(
                        new Identifier("file-1")).uri(
                        URI.create(TestUtil.class.getClassLoader().getResource(
                                "scape_logo.png").toString())).technical(
                        TestUtil.createMIXRecord()).build();

        Representation rep =
                new Representation.Builder(new Identifier("representation-1"))
                        .files(Arrays.asList(f)).title("Text representation")
                        .technical(TestUtil.createTextMDRecord()).provenance(
                                TestUtil.createPremisDigiProvRecord()).rights(
                                TestUtil.createPremisRightsRecord()).source(
                                TestUtil.createDCSourceRecord()).build();

        IntellectualEntity e =
                new IntellectualEntity.Builder().identifier(
                        new Identifier(entityId)).representations(
                        Arrays.asList(rep)).descriptive(
                        TestUtil.createDCRecord()).build();
        return e;
}
```

[19] https://github.com/openplanets/scape-platform-datamodel

6 The Connector API

The Connector API's purpose in the SCAPE platform is to integrate different repositories with the various SCAPE components. As such it sits on top of the content repository and exposes well defined services via HTTP used by clients to access the repository content. Clients can use the proposed Connector API to access not only content but also preservation plans.

When preserving collections with mainly small individual content, the overhead to request binary content via HTTP can be neglected, but when requesting large binary content from the repository via HTTP the overhead gets significant in respect to the request duration. Therefore two storage strategies are introduced that handle binary content differently: one is responsible for the whole lifecycle of a binary object, the second storage strategy only handles references to binary content, while an outside agent is responsible for content manipulation.

The Connector API is required to accommodate the various use cases in a SCAPE platform arising from the different API clients. The computation cluster as a client of the Connector API must be able to perform CRUD[20] operations on digital objects, while preservation plan management requires CRUD operations on preservation plans, and the discovery of digital objects via SRU[21] searches is required by, e.g., plan experimenting.

The Connector API is, aside from the Report API, one of the two APIs a repository has to implement in order to be employed in a SCAPE platform. Therefore arbitrary repository systems can be used in a SCAPE environment, if and only if they implement the Connector and the Report API.

Since well-defined requests have to be used by clients of the Connector API, the format of the objects is part of the API definitions. The data model definitions for the SCAPE platform has been discussed in this document.

6.1 Use Cases of the Connector API

Loader application for batch ingest
The Loader Application will be a client application of the repository that enables the administrator to ingest data into the repository in a batch process. The Loader application takes care of validation, error logging and retry functionality. The details of the requirements will be described in a separate document. The loader application requires a HTTP endpoint for ingesting Intellectual Entities in the repository.

Request Intellectual Entities by the computation cluster
In order to run actions on Intellectual Entities for characterization, identification and other tasks and depending on the Storage Strategy as described in Section 6.2 the data or their references are needed for task execution. References to the data can be extracted from Intellectual Entities requested. Content can also be requested directly from the repository by using an Identifier as described and - depending on the Storage Strategy - the repository returns the content directly or uses HTTP redirection to route the request to the content's location.

Update Intellectual Entities with provenance information
To enable preservation actions executed on the computation cluster updating provenance metadata of Representations an interface is needed. Preservation actions have to employ the HTTP endpoint to

[20] http://en.wikipedia.org/wiki/Create,_read,_update_and_delete
[21] http://www.loc.gov/standards/sru/

update Intellectual Entities, although it may be possible to expose an endpoint for updating single representations in the future. If e.g. the computation cluster updated an image file and provenance information had to be written into the corresponding Representation, the information had to be updated in the repository by using the HTTP endpoint to update the whole Intellectual Entities.

Partially fetching large scale files
A repository may hold Files such as ARC containers large in size. Since preservation tasks may only be concerned with a subset of the File, fetching the whole container would be uneconomic and have negative impact on performance. Therefore the requirement to partially request Files arises.

6.2 Storage Strategy

The heterogeneity of collections in regard to size poses a challenge for a well performing implementation: while fetching small collections through an API exposed via HTTP for workflow execution is a practical approach, the performance to fetch very large collections will be poor. Therefore the following two Storage Strategies are available, letting stakeholders choose the more fitting alternative:

1. *Managed Content:* When using this strategy, Files are accessible only via the Connector API, so that any operations involving Files the binary data requires the clients to use the Connector API to retrieve or update the binary data from the repository. This functionality is exposed to the clients by an HTTP endpoint. This strategy is not very well suited for large amounts of data or geographically separated storage and computation clusters, because of the necessary I/O overhead for data retrieval by the computation cluster.
2. *Referenced Content:* When using this strategy, the repository is integrated with the different components by storing Files in a file system directly accessible by the SCAPE platform, and only referencing Files in the repository by an URI. This enables the platform to handle large files without having to move them in between machines before computation can take place.

The first approach enables preservation actions to be performed on an external computation cluster, which is in use for a limited time only. The export of the data to the cluster is performed via the Connector API. In this scenario the computation cluster and storage are physical and/or geographical distinct entities. Drawbacks to this approach are that an institutions policy might prevent exporting the data to an untrusted third party and it introduces a bottleneck by exporting the data over a network — a worst-case scenario would be to move the entire content — to a computation cluster.
The second approach is more eligible for running a Hadoop cluster with access to the same HDFS file system the Files are stored in. Computation Cluster and Storage are not distinct instances in this scenario, requiring the hardware to act as a storage and computation platform for the preserved Intellectual Entities.

6.3 Schematic views

Fetching an Intellectual Entity

To fetch an Intellectual Entity from the repository via the Connector API the following workflow has been identified:

1. A Taverna workflow[22] has been scheduled for execution on the computation cluster
2. A process on the computation cluster requests collection metadata from the repository via the Connector API
3. The repository gathers the Intellectual Entities' metadata from the metadata storage
4. The repository answers the process' Connector API request by returning the XML representation for the Intellectual Entity.
5. If referenced content is used the computation cluster can access the binary data by resolving URIs in the collection profile, otherwise the computation cluster requests the binary data from the repository via the Connector API.

The following sequence diagram illustrates this workflow to retrieve an Intellectual Entity from the repository.

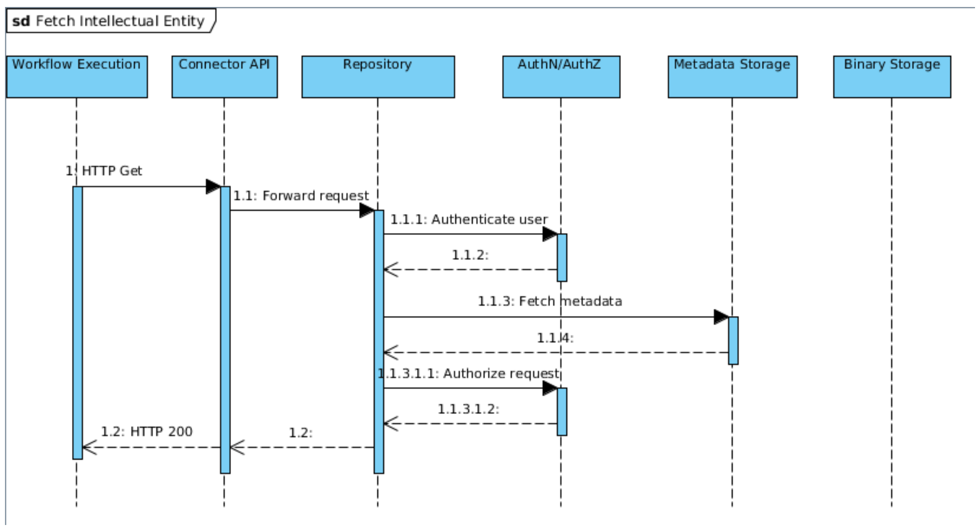


Figure 6-1: Sequence Diagram illustrating the workflow to fetch an Intellectual Entity

Fetching a File

This two sequence diagrams illustrate the workflow to fetch a File from the repository in a scenario where the files are handled as referenced content and in the second diagram within a managed content scenario.

[22] http://www.taverna.org.uk/

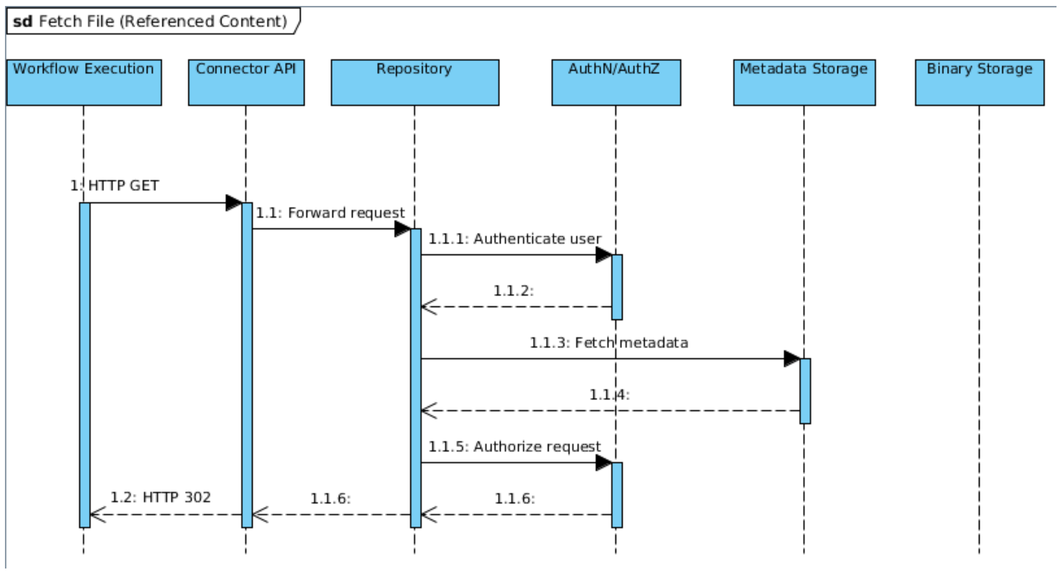


Figure 6-2: Sequence Diagram illustrating the workflow to fetch a File (as referenced content)

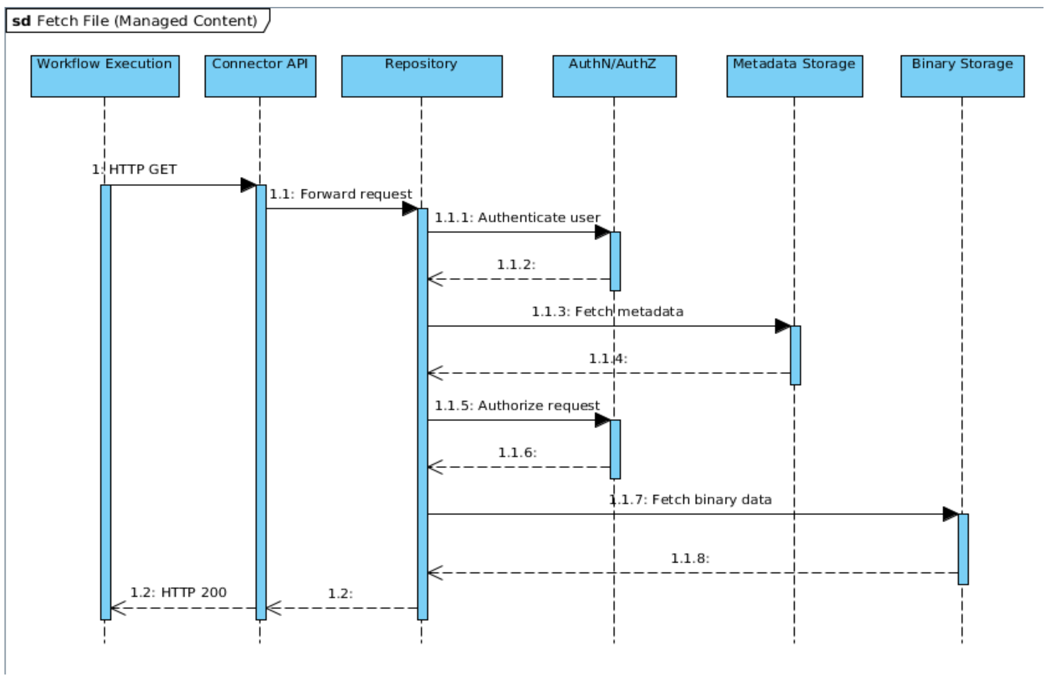


Figure 6-3: Sequence Diagram illustrating the workflow to fetch an Intellectual Entity (as managed content)

Ingesting or Updating an Intellectual Entity

To ingest or to update an Intellectual Entity in a repository via the Connector API the following workflow has been identified:

1. A Taverna workflow has executed successfully and the resulting Intellectual Entities have to be written back to the repository.
2. A process on the computation cluster sends the updated Intellectual Entity via the Data Connector API to the repository, and references the updated binary data in the request.
3. The repository updates the Intellectual Entities' metadata, and if using managed content the updated binary data gets written from the referenced location to the binary storage.
4. The repository answers the process' request and informs about any errors that might have occurred while the operation was performed.

The following sequence of diagrams illustrates this workflow for two storage scenarios of managed and referenced content:

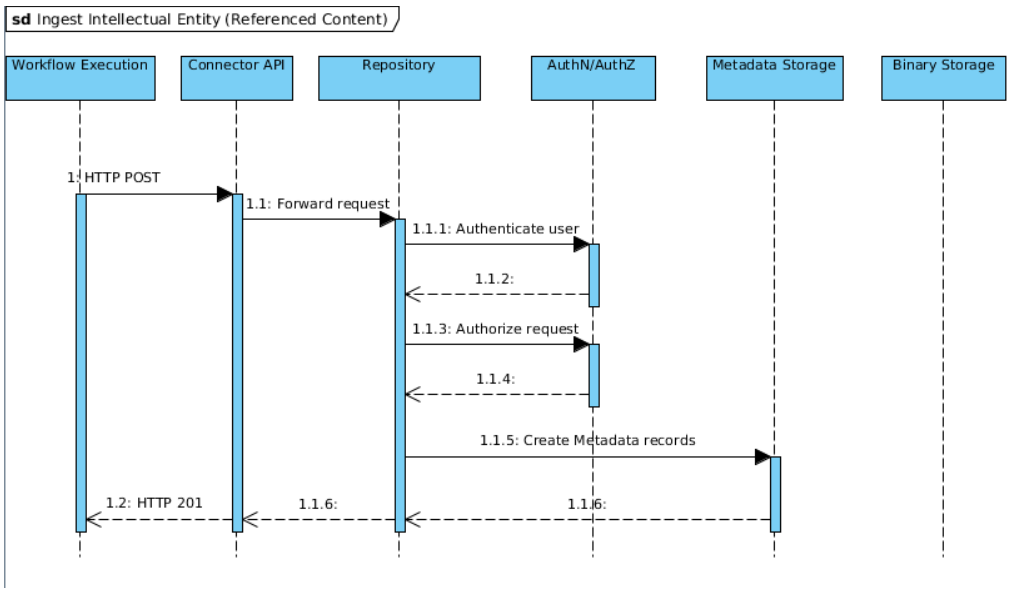


Figure 6-4: Sequence Diagram illustrating the workflow to ingest an Intellectual Entity (as referenced content)

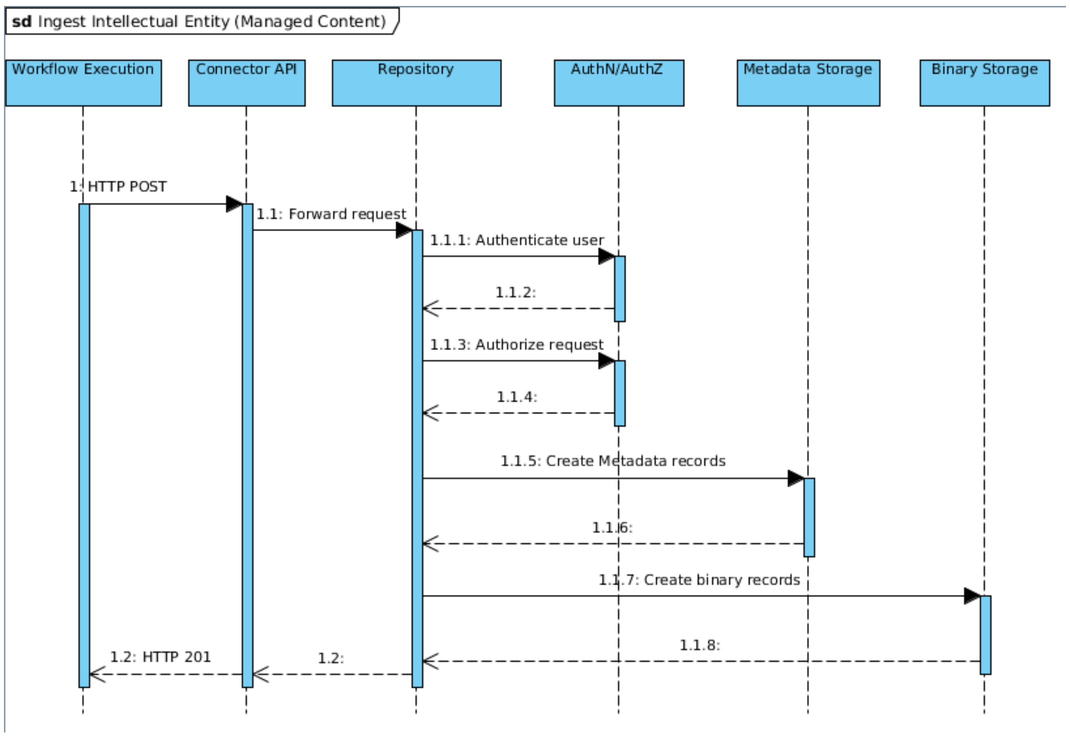


Figure 6-5: Sequence Diagram illustrating the workflow to ingest an Intellectual Entity (as managed content)

7 Specification of the Connector API

The content repository exposes a RESTful[23] API to the workflow execution layer on top of a Hadoop computation cluster, for operations on the metadata of Intellectual Entities. The API consists of various HTTP endpoints defined in the following section.

The preserved Intellectual Entities consist of administrative, structural, rights and descriptive metadata and associated Files and are themselves versioned. Existing and well-established standards for encoding the different aspects of metadata are used for wrapping the metadata of Intellectual Entities.

7.1 Authentication & Authorization

Following the REST recommendations authentication is done by using Basic and Digest Access Authentication mechanism, also known as HTTP Basic Authentication, which is well established and supported throughout the otherwise heterogeneous IT landscape. This implies that every HTTP request has to have a BASE64 encoded user name and password string in the Authentication Header. Therefore encryption by using HTTP over SSL/TLS is strongly recommended.

The repository does authorization depending on the current user and the individual Representations' associated rights and permission metadata.

7.2 Life cycle states

The life cycle state of an Intellectual Entity contains the information describing the state of an Intellectual Entity in the preservation lifecycle. Currently three states are defined: INGESTED, INGEST_FAILED, OTHER. Each of these states also has details associated with it, so the repository can supply additional information. These states result from the following use case: When ingesting SIPs asynchronously a user has to be able to get information about the ingestion process: Whether ingestion succeeded, failed or is currently in process. In order to supply information about currently running ingestion processes (e.g. virus checking) "OTHER" can be used with it's details describing the process of the ingestion workflow the SIP is currently in.

```
<lifecyclestate id="entity-42" state="INGESTED">
       <details>Ingest finished successfully at 6/27/2012 13:34:57</details>
</lifecyclestate>
```

Example 1: XML Representation of the life cycle state of an ingested Intellectual Entity

7.3 HTTP Status codes

The existing HTTP status codes with their individual semantics are used in the context of the connector API.

- **200 OK** This indicates success.
- **201 Created** means that an object has been created in the repository.
- **401 Unauthorized** The request requires authentication. The user should authenticate properly against the repository and repeat the request.
- **403 Forbidden** The server refuses to fulfil the request. Authorization will not help and the request should not be repeated.

[23] R.T. Fielding, 2000,"Architectural Styles and the Design of Network-based Software Architectures"

- **404 Not Found** The requested resource cannot be found.
- **415 Unsupported Media Type** The media type sent with the requested was not valid**.**
- **500 Internal Server Error** In the case of runtime errors that might be happening while requesting a resource: e.g., Disk full.

7.4 HTTP endpoints

Following is a specification of the HTTP endpoints. The implementation of the endpoints has to be done by each repository in order to be deployed in a SCAPE Platform environment.

The XML Schemas of the XML responses are defined by the Scape Digital Object Model and by metadata frameworks like Dublin Core and VideoMD. The schemas for object representations as defined in the Digital Object Model are generated by the JAX-B implementation's schemagen tool. The schemas for existing metadata frameworks employed by the SCAPE platform are readily available on the web.

Title	*Retrieve an Intellectual Entity*
Comment	Retrieval of entities is done via a GET request. Since Intellectual Entities can have multiple versions there is an optional version identifier, which when omitted defaults to the most current version of the Intellectual Entity. When successful the response body is a METS representation of the Intellectual Entity. The parameter *useReferences* controls whether the response is created using references to the metadata via <mdRef> elements or if the metadata should be wrapped inside <mdWrap> elements in the METS document.
Path	/entity/<entity-id>/<version-id>?useReferences=[yes|no]
Method	HTTP/1.1 GET
Parameters	entity-id: *the id of the requested Intellectual Entity* version-id: t*he version of the requested entity (optional)* useReferences: An optional parameter that says *whether to wrap metadata inside <mdWrap> elements ("no") or to reference the metadata using <mdref> elements ("yes"). Defaults to "yes".*
Produces	A XML representation of the requested Intellectual Entity version
Content Type	text/xml

Title	***Retrieve a set of Intellectual Entities***
Comment	Retrieval of single metadata records of entities is done via a GET request. Since Intellectual Entities can have multiple versions there is an optional version identifier, which when omitted defaults to the most current version of the Intellectual Entity. When successful the

	response body is a XML representation of the corresponding metadata record.
Path	/metadata/<entity-id>/<rep-id>/<file-id>/<bitstream-id>/<version-id>/<md-id>
Method	HTTP/1.1 GET
Parameters	entity-id: *the id of the Intellectual Entity* rep-id: *the id of the Representation (optional)* file-id: *the id of the File (optional)* bitstream-id: *the id of the requested binary content (optional)* *md-id: the id of the metadata to retrieve* version-id: t*he version of the requested bit stream's parent Intellectual Entity (optional)*
Produces	A XML representation of the requested metadata record according to the corresponding metadata's schema
Content Type	text/xml

Title	*Retrieve a metadata record*
Comment	In order to make fetching a whole set of entities feasible, this GET method consumes a list of URIs sent with the request. It resolves the URIs to Intellectual Entities and creates a response consisting of the corresponding METS representations. If at least one URI could not be resolved the implementation returns a HTTP 404 Not Found status message.
Path	/entity-list
Method	HTTP/1.1 POST
Parameters	A text/uri-list of the entities to be retrieved
Produces	METS representations of the requested entities.
Content Type	multipart

Title	*Ingest an Intellectual Entity*
Comment	Ingestion of digital objects is done by sending a METS

	representation of an Intellectual Entity in the body of a HTTP POST request, which gets validated and persisted in the repository. If validation does not succeed the implementation returns a HTTP 415 "Unsupported Media Type" status message. When successful the response body is a plain text document consisting of the ingested entity's identifier.
Path	/entity
Method	HTTP/1.1 POST
Parameters	N/A
Consumes	A XML representation of the entity
Produces	The Intellectual Entity identifier
Content Type	text/plain

Title	*Ingest an Intellectual Entity asynchronously*
Comment	Ingestion is done by sending a SIP to this endpoint. The method returns immediately and supplies the User with an ID that can be used to request the status of the ingestion.
Path	/entity-async
Method	HTTP/1.1 POST
Parameters	N/A
Consumes	A XML representation of the entity
Produces	An Identifier. The identifier can be used to request the lifecycle status of the digital object ingested.
Content Type	text/plain

Title	*Update an Intellectual Entity*
Comment	In order to allow updating of Intellectual Entities, the implementation exposes this HTTP PUT endpoint. The mandatory parameter <id> tells the repository which Intellectual Entity is to be updated. The request must include the updated METS representation of the entity in the request body.

Path	/entity/<id>
Method	HTTP/1.1 PUT
Parameters	Id: *the id of the Intellectual Entity to update*
Consumes	A digital object's XML representation.
Produces	the id of the entity
Content Type	*text/xml*

Title	*Retrieve a version list for an Intellectual Entity*
Comment	In order to get a list of all versions of an Intellectual Entity a plain GET request can be sent to the implementation with the <id> parameter indicating which entity's versions to list. If successful the response consists of the Intellectual Entity's version identifiers in a XML document.
Path	/entity-version-list/<entity-id>
Method	HTTP/1.1 GET
Parameters	entity-id: *the id of the Intellectual Entity*
Consumes	N/A
Produces	A XML representation of all the entities version ids.
Content Type	*text/xml*

Title	*Retrieve a File*
Comment	For fetching Files associated with Intellectual Entities, the implementation exposes a HTTP GET endpoint. Requests sent to this endpoint must have a <id> parameter indicating which File to fetch. The parameter <version-id> indicating the version to fetch is optional and defaults to the most current version of the File. Depending on the Storage Strategy the response body is the binary file with the corresponding Content-Type set by the repository or a HTTP 302 redirect in the case of referenced content.
Path	entity-id: *the id of the Intellectual Entity* representation-id: *the id of the Representation*

	file-id: *the id of the File* version-id: t*he version of the requested File's parent Intellectual Entity (optional)*
Method	HTTP/1.1 GET
Parameters	entity-id: *the id of the Intellectual Entity*
Consumes	N/A
Produces	the file requested or a redirect to the file when using referenced content.
Content Type	depends on File's metadata, but defaults to application/octet-stream.

Title	*Retrieve named bit streams*
Comment	For fetching a named subset of Files, such as an entry in an ARC container, the implementation exposes a HTTP GET method. The mandatory parameter <id> is the identifier of the requested bit stream in the Intellectual Entity. Depending on the Storage Strategy the implementation returns the bit stream directly in the response body, or it redirects the request using HTTP 302 to the referenced content. This requires special care when using Referenced Content as a Storage Strategy since the implementation is only able to redirect to referenced bit streams, making the redirect target responsible for answering the request properly.
Path	/bitstream/<entity-id>/<rep-id>/<file-id>/<bitstream-id>/<version-id>
Method	HTTP/1.1 GET
Parameters	entity-id: *the id of the Intellectual Entity* rep-id: *the id of the Representation* file-id: *the id of the File* bitstream-id: *the id of the requested binary content* version-id: t*he version of the requested bit stream's parent Intellectual Entity (optional)*
Consumes	N/A
Produces	The binary content associated requested, or a redirect to the binary

	content.
Content Type	Depends on content's type, but defaults to application/octet-stream.

Title	*Search Intellectual Entities in a collection*
Comment	For digital object discovery, the implementation exposes a SRU search endpoint. The endpoint implements the SRU specifications by the Library of Congress for Internet Search queries, utilizing CQL, a standard syntax for representing queries, and exposes this functionality via a HTTP GET endpoint. Pagination is done via the SRU parameters *startRecord* and *maximumRecords*[24]
Path	/sru/entities
Method	HTTP/1.1 GET
Parameters	see SRU specification[25]
Consumes	N/A
Produces	A XML representation as specified by SRU
Content Type	text/xml

Title	*Search Representations in a collection*
Comment	For discovering Representations, the implementation exposes a SRU search endpoint. The endpoint implements the SRU specifications by the Library of Congress for Internet Search queries, utilizing CQL, a standard syntax for representing queries, and exposes this functionality via a HTTP GET endpoint. Pagination is done via the SRU parameters *startRecord* and *maximumRecords*
Path	/sru/representations
Method	HTTP/1.1 GET
Parameters	see SRU specification[26]

[24] http://www.loc.gov/standards/sru/specs/search-retrieve.html

[25] http://www.loc.gov/standards/sru/

Consumes	N/A
Produces	A XML representation as specified by SRU
Content Type	text/xml

Title	*Search Files in a collection*
Comment	For discovering Files, the implementation exposes a SRU search endpoint. The endpoint implements the SRU specifications by the Library of Congress for Internet Search queries, utilizing CQL, a standard syntax for representing queries, and exposes this functionality via a HTTP GET endpoint. Pagination is done via the SRU parameters *startRecord* and *maximumRecords.*
Path	/sru/files
Method	HTTP/1.1 GET
Parameters	see SRU specification[27]
Consumes	N/A
Produces	A XML representation as specified by SRU
Content Type	text/xml

Title	*Retrieve the lifecycle status of an entity*
Comment	In order to access the life cycle state of an Intellectual Entity without having to fetch the whole METS representation, an endpoint for retrieving this significant property is exposed by the repository.
Path	/lifecycle/<entity-id>
Method	HTTP/1.1 GET
Parameters	entity-id: *the id of the Intellectual Entity to update*
Consumes	N/A

[26] http://www.loc.gov/standards/sru/
[27] http://www.loc.gov/standards/sru/

Produces	A XML representation of the lifecycle status
Content Type	text/xml

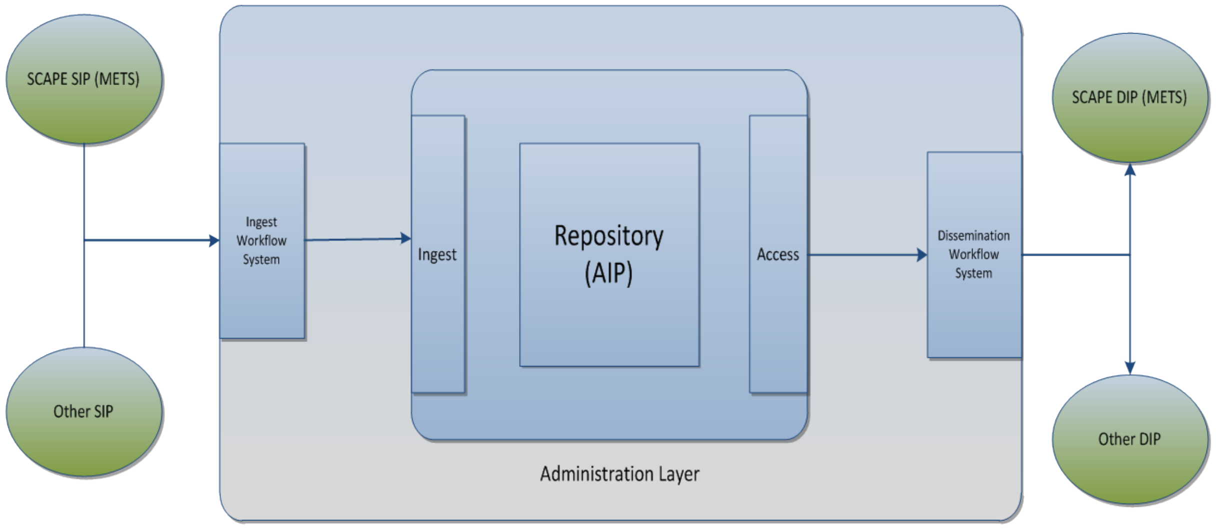
Title	*Retrieve a Representation*
Comment	For fetching Representations without having to retrieve the METS representation of the whole Intellectual Entity, a dedicated endpoint is exposed by the repository
Path	/representation/<entity-id>/<representation-id>
Method	HTTP/1.1 GET
Parameters	entity-id: *the id of the Intellectual Entity to update* representation-id: *the id of the Representation to update*
Consumes	N/A
Produces	A XML representation of the requested Representation
Content Type	text/xml

Title	*Update a Representation of an Intellectual Entity*
Comment	For updating a Representation of an Intellectual entity without sending a METS representation of the Intellectual Entity, an endpoint is exposed by the repository. The repository *has to* create a new Version of the Intellectual Entity with the updated Representation.
Path	/representation/<entity-id>/<representation-id>
Method	HTTP/1.1 PUT
Parameters	entity-id: *the id of the Intellectual Entity to update* representation-id: *the id of the Representation to update*
Consumes	A Representation’s XML representation.
Produces	the entity-id and representation-id
Content Type	text/xml

Title	*Update the metadata of an Intellectual Entity*
Comment	When updating only the metadata of an Intellectual Entity, validity checking on binary files can be omitted, thereby saving a substantial number of CPU cycles. An endpoint is exposed to clients for updating the metadata of an Intellectual entity; it consumes a METS representation of an Intellectual Entity.
Path	/metadata/<entity-id>/<metadata-id>
Method	HTTP/1.1 PUT
Parameters	entity-id: *the id of the Intellectual Entity to update* metadata-id: *the Id of the metadata set to update*
Consumes	A metadata set's XML representation.
Produces	the entity-id an metadata-id
Content Type	text/xml

8 Conclusion

We described a Digital Object Model of SCAPE based on METS and PREMIS. We explained a METS profile that must be used by all SCAPE partners in order to prevent a situation where a partner cannot integrate with SCAPE because his data model is something totally different. Beside that we defined the OAIS Information packages used by the repositories and other systems or users that interact with the repository (e.g., SCAPE Watch component). We also discussed the public available implementation of the SCAPE Digital Object model and outlined the specification and use cases of the Data Connector API, which is a mandatory interface for all DORs that integrate with the SCAPE platform.



9 Glossary

Representation (PREMIS term)
The set of files, including structural metadata, needed for a complete and reasonable rendition of an Intellectual Entity. For example, a journal article may be complete in one PDF file; this single file constitutes the representation. Another journal article may consist of one SGML file and two image files; these three files constitute the representation. A third article may be represented by one TIFF image for each of 12 pages plus an XML file of structural metadata showing the order of the pages; these 13 files constitute the representation. From Introduction and Supporting Materials from PREMIS Data Dictionary, p. 7.[28]

Metadata
Metadata is information about an analogue or digital object, a component of an object, or a coherent collection of objects. Metadata describing digital content is often structured (e.g., with tagging or markup) and it may be embedded within a single file, incorporated within the "packaging" that is associated with a group of files (e.g., METS), placed in a related external file (e.g., XMP sidecar file), or in a system external to the digital file (e.g., a database) to which the digital file or files are linked via a unique key or association.[29]

Metadata, administrative
Administrative metadata is metadata that is used for the management of digital content, such as information about rights and permissions (see Metadata, rights) as well as other facts about a given digital object. Some speakers define *administrative metadata* to include technical metadata (see Metadata, technical), source metadata (see Metadata, source), and process metadata (see Metadata, process).[30]

Intellectual entity (PREMIS term)
A set of content that is considered a single intellectual unit for purposes of management and description: for example, a particular book, map, photograph, or database. An Intellectual Entity can include other Intellectual Entities; for example, a Web site can include a Web page; a Web page can include an image. An Intellectual Entity may have one or more digital representations. From Introduction and Supporting Materials from PREMIS Data Dictionary, p. 6.[31]

File (PREMIS term)
File is a named and ordered sequence of bytes that is known by an operating system. A file can be zero or more bytes and has a file format, access permissions, and file system characteristics such as size and last modification date. Files can be read, written, and copied. Files have names and formats." *Introduction and Supporting Materials from PREMIS Data Dictionary* (p. 7)[32]

[28] http://www.digitizationguidelines.gov/term.php?term=representation
[29] http://www.digitizationguidelines.gov/term.php?term=metadata
[30] http://www.digitizationguidelines.gov/term.php?term=metadataadministrative
[31] http://www.digitizationguidelines.gov/term.php?term=intellectualentity
[32] http://www.digitizationguidelines.gov/term.php?term=digitalfile

Bitstream
A bitstream is contiguous or non-contiguous data within a file that has meaningful common properties for preservation purposes. Generally speaking, a bitstream cannot be transformed into a standalone file without the addition of file structure (headers, etc.) and/or reformatting the bitstream to comply with some particular file format. This definition is derived from the data model outlined in *Introduction and Supporting Materials from PREMIS Data Dictionary*, p. 7, illustrated by the example of a TIFF file that contains embedded bitstreams representing raster images together with header that presents some information about the file. The authors of the PREMIS definition note that their definition is limited to sets of bits embedded within a file and they call attention to an alternate usage that defines *bitstream* as an entity that could span more than one file.[33]

Information Package
Conceptual linking of Content Information plus Preservation Description Information plus Packaging Information (Submission, Archival and Dissemination Information Packages).

Archival Information Package (AIP)
An AIP is an Information Package, consisting of the Content Information and the associated Preservation Description Information (PDI), which is preserved within an OAIS.[34]

Dissemination Information Package (DIP)
A DIP is an Information Package, derived from one or more AIPs, received by the Consumer in response to a request to the OAIS.[35]

Submission Information Package (SIP)
A SIP is an Information Package that is delivered by the Producer to the OAIS for use in the construction of one or more AIPs.[36]

Apache Hadoop
Hadoop is a software framework that supports data-intensive distributed applications under a free license. It enables applications to work with thousands of computational independent computers and petabytes of data. Hadoop was derived from Google's MapReduce and Google File System (GFS) papers.

Technology Compatibility KIT
A TCK is a suite of tests that at least nominally checks a particular alleged implementation for compliance.

Metadata Encoding and Transmission Standard (METS)
METS is a metadata standard for encoding descriptive, administrative, and structural metadata regarding objects within a digital library, expressed using the XML schema language of the World Wide Web Consortium. The standard is maintained in the Network Development and MARC Standards Office of the Library of Congress, and is being developed as an initiative of the Digital Library Federation.

[33] http://www.digitizationguidelines.gov/term.php?term=bitstream
[34] Reference Model for an Open Archival Information System (OAIS), CCSDS 650.0-B-1 BLUE BOOK
[35] Reference Model for an Open Archival Information System (OAIS), CCSDS 650.0-B-1 BLUE BOOK
[36] Reference Model for an Open Archival Information System (OAIS), CCSDS 650.0-B-1 BLUE BOOK

Representational state transfer (REST)
REST is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Fielding is one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1.

Contextual Query Language (CQL)
CQL, previously known as Common Query Language is a formal language for representing queries to information retrieval systems such as search engines, bibliographic catalogues and museum collection information. Based on the semantics of Z39.50, its design objective is that queries be human readable and writable, and that the language be intuitive while maintaining the expressiveness of more complex query languages. It is being developed and maintained by the Z39.50 Maintenance Agency, part of the Library of Congress.

Search/Retrieve via URL (SRU)
SRU is a standard search protocol for Internet search queries that uses Contextual Query Language (CQL) as a query syntax for representing queries.

10 List of figures