**Fall**

16

**DATA ARCHIVING AND NETWORKED SERVICES (DANS)**

EASY Architecture Guide v2.8

Jan van Mansum

08

**Fall**

# Table of Contents

Table of Contents 3

0 Introduction 5

Purpose and Audience 5

Views 5

1 Conceptual View 7

Overview 7

Components 8

2 Functional View 11

Overview 11

EASY Object Model 11

Use cases 13

3 Process View 16

Overview 16

Processes and Protocols 17

Example Interactions 19

4 Software View 21

Overview 21

Programs 22

5 Digital Preservation View 25

Overview 25

OAIS Information Model 25

Fedora Commons Digital Object Model 27

Digital Preservation in EASY 28

References 29

# 0 Introduction

This document describes the architecture of the DANS EASY system.

## Purpose and Audience

The Electronic Archiving SYstem (EASY) is a Fedora Commons based digital archiving system for the long term preservation of scientific research data. This document is a guide to the architecture of EASY. It describes the existing architecture of version 2.8. It is intended as a basis for discussion about future development of EASY.

views

The approach taken is to describe the system from several different angles, called *views*. Each view discusses the system as a whole, but limits itself to certain aspects.

The architecture of EASY may be of interest to many people in different roles and with different areas of expertise, such as customers of DANS, developers of EASY, IT staff maintaining the servers running the system, DANS management, etc. The views presented below are of varying relevance to each of these groups.

## Views

The following views have been elaborated, each in its own module.

### Conceptual

The *conceptual view* gives an overview of the main compontents of the system, their roles and relations. It also gives examples of some types of users of the system. The conceptual view is a good starting point for learing about the architecture of EASY.

### Functional

The *functional view* describes what high-level use cases the system supports. It does *not* go into detail about how these use cases are implemented. Apart from the use cases this view discusses the conceptual interface that is presented to users of the system. The functional view is important for users of EASY, both DANS staff and external users. It is also essential reading for members of the EASY development team.

### Process

The *process view* examines the processes that make up a running EASY system. It describes how these processes communicate with each other. There is a strong resemblance with the conceptual view, but it goes into a bit more detail about the technical implementation. It is important for EASY software developers and ICT Support staff who are involved in setting up and maintaining the platform on which EASY runs.

### Software

The *software view* shows how the system is implemented in terms of sofware components. What components are developed at DANS? How are they divided up in modules? What are the dependencies between the modules? This view is important mainly for EASY software developers.

### Digital Preservation

The *digital preservation view* is especially important. While the above views are commonly employed in architectural descriptions, the digital preservation view is particular to EASY’s problem domain. Ultimately, the goal of EASY is to preserve information for the long term. This requires us to deal with a series of challenges ranging from preserving the data bits as they have been ingested to making sure that the information encoded in those bits can be restored in the minds of the Designated Community (even a long way into the future). The digital preservation view details how EASY is designed to provide these guarantees. In the discussion the OAIS is used as a reference model. It is essential reading for (technical) archivists and information specialists.

# 1 Conceptual View

The conceptual view describes:

1. the main logical components of the system;
2. the relations between these components;
3. the types of users that can interact with the system.

## Overview

EASY consists of a number of cooperating components.

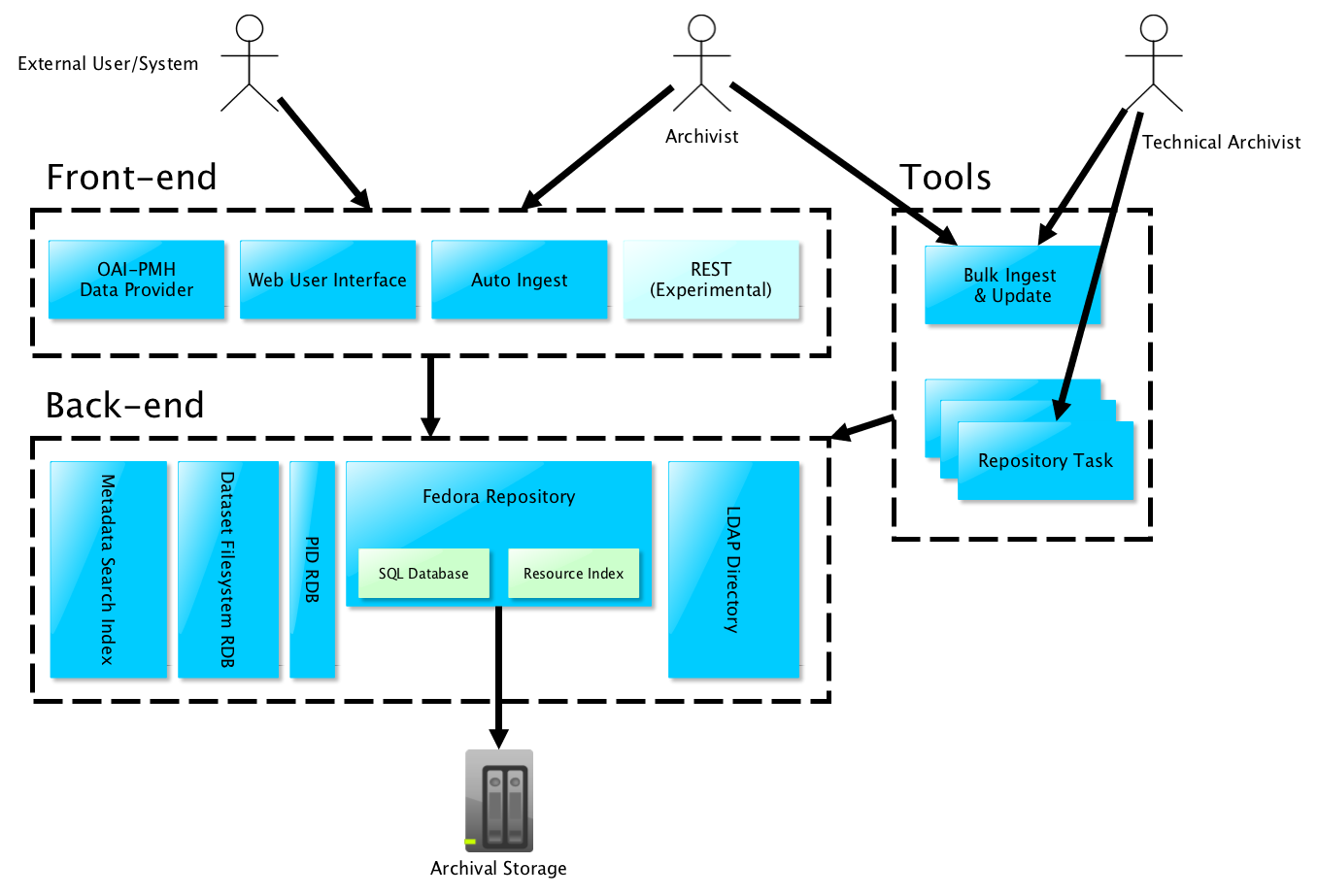


Figure 1 EASY Conceptual Overview

*front-end versus back-end*

A first distinction to make is the one between front-end and back-end components. The front-end components are the ones that interact with users of the system. The back-end components implement the majority of the services offered by the system.

*tools*

Apart from front-end and back-end there are also tools that are similar to front-end components, but allow more direct access to the core of EASY. Tools are intended for use by privileged users only. At DANS privileged users are (technical) archivists and application managers.

Users of the system may also be other systems. This is the case for example with the OAI-PMH Data Provider. It is accessed by so-called OAI-PMH Service Providers, software programs that “harvest” the metadata in the EASY repository by means of the Data Provider.

Figure 1 EASY Conceptual Overview summarizes the description given in the previous paragraphs. The arrows indicate that an actor or component has access to the target. Data can still flow in both directions. Archival Storage is placed outside the back-end group to highlight the fact that only Fedora has direct access to it.

## Components

What follows is a brief description of each of the conceptual components.

### Archival Storage

We begin our exploration of the EASY system at the bottom of Figure 1. The Archival Storage component is a storage component that contains the research data in a form suitable for long term preservation. This will be explained in more detail in the *Section* 5 Digital Preservation View.

### Back-end

#### Fedora Commons Repository

At the very core of EASY is the Fedora Commons Repository. This component manages the Archival Storage. In more concrete terms: this component makes decisions about the contents, names and location of the files in Archival Storage. It is the gateway through which the front-end components and tools access the data stored in the archive.

Since its main goal is long term preservation the Archival Storage is not necessarily laid out in a way that supports fast look-up and retrieval of data. For this purpose Fedora maintains two “indexes” into the Archival Storage:

1. An SQL Database that maps Fedora IDs to locations in Archival Storage.
2. A Resource Index, which answers queries about the RDF statements that can be stored in the repository.

The information in these two indexes is redundant, i.e. it is also present in Archival Storage and can be rebuilt from the information in Archival Storage. The SQL database and Resource Index are part of the Fedora software, that is why they are represented as blocks inside the Fedora block in Figure 1.

#### LDAP Directory

Data about users of EASY is not stored in Archival Storage. Instead, it is contained in an LDAP directory. The LDAP directory also stores mappings of EASY 1 identifiers to EASY 2 (Fedora) identifiers.

#### Dataset Filesystem RDB

EASY organizes the files it archives conceptually as “datasets”, i.e. collections of files that belong together. Datasets are presented to the user as filesystem-like tree of files and folders (see *Section* 2 Functional View). Datasets are however *not* stored as actual filesystem directories in Archival Storage. Instead, the location of the various parts of a dataset is maintained by Fedora. To support efficient look-up EASY maintains its own⎯redundant⎯mapping of dataset parts to Fedora objects in the form of the Dataset Filesystem RDB (Relational Database).

#### Metadata Search Index

To support discovery of datasets EASY maintains a Metadata Search Index. As the name implies it lets the user search for datasets by means of queries about the metadata stored with the dataset.

### Front-end

#### Web User Interface

Web-UI

The most important front-end component is the Web User Interface (or Web-UI for short). It is a web-based application that supports ingest of datasets by users, processing, publishing and maintaining datasets by archivists and discovery and download of (parts of) datasets by users.

#### OAI-PMH Data Provider

The OAI-PMH is a protocol that lets Service Providers harvest metadata from Data Providers. EASY can function as an OAI-PMH Data Provider.

#### Auto-Ingest

EASY also has a machine-machine interface for ingesting datasets into the archive. This interface is called Auto-Ingest, short for “Automated Ingest”.

#### REST (Experimental)

The REST service is an experimental RESTful service that supports retrieval of metadata and file data, searching …

<checken by Georgi wat precies de scope is>

This component is experimental because it was conceived as an experiment in using a RESTful interface in EASY. That is why it has a different shade of blue in Figure 1. The name reveals the lack of functional focus. In the future it should probably be broken up in several RESTful services, each with a clearly defined purpose.

### Tools

Tools are used only by (technical) archivist and provide a lower level access to EASY.

#### Bulk Ingest & Update (EBIU)

EBIU

The Bulk Ingest & Update tools allows an archivist to ingest datasets en masse or to add or update metadata to datasets file items in datasets.

#### Repository Tasks

Repository Tasks are specialized programs that perform various low-level tasks, some of which are scheduled to be run automatically on a daily or weekly basis. Examples are:

1. Export of meta-data and statistics for the Data Management System (<check: is dit de naam?>)
2. Assigning datasets to collections

# 2 Functional View

The functional view describes:

1. the abstract concepts that a user works with;
2. the use cases supported by the system in terms of these concepts.

## Overview

There are many possible users of EASY; they can be human or software clients, they can be DANS internal or external. What they have in common is that they work with the abstractions that EASY offers to them. We call those abstractions the EASY Object Model.

The types of things they can do with the abstractions provided in the EASY Object Model can be described in use case scenarios.

We will now first give an overview of the EASY Object Model and then enumerate the use cases supported by the system.

## EASY Object Model

### Datasets

The EASY Object Model centers around the notion of a dataset. A dataset in EASY is a collection of file items, with common metadata, organized in a directory-tree.

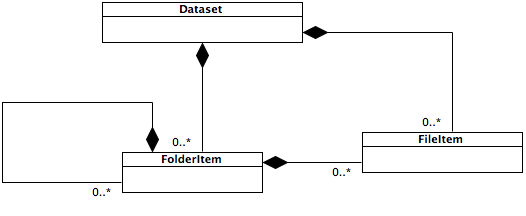


Figure 2 EASY Dataset Structure

Figure 2 EASY Dataset Structure expresses that datasets contain folders and files and that folders contain files and other folders. The filled diamond shape denotes that the corresponding side of the association is the container of the other side. In other words, if the side marked with the diamond is deleted, the contained elements are also deleted[[1]](#footnote-1). For an overview of the UML-notations used in this document, see …< UML cheat sheet>

### Collections

Datasets can also be grouped by defining “collections”. See Figure 3 EASY Collections. This type of grouping is looser than the grouping of file and folder items into datasets. If a collection is deleted, the datasets that belonged to it remain. A dataset can belong to more than one collection and a collection can contain other collections.

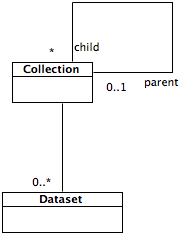


Figure 3 EASY Collections

### Users

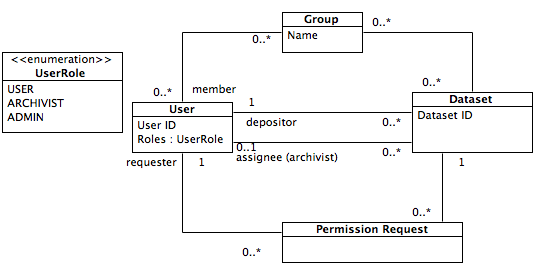


Figure 4 EASY Users

Figure 4 EASY Users displays that users themselves are also represented in the system.

A user can be linked to a dataset in several ways:

1. the user is the depositor of the dataset;
2. the user (an archivist) is assigned a dataset to process it;
3. the user has requested to download files from a dataset that requires permission from the depositor;
4. the user is member of one of the groups that has access to the group level file items of the dataset.[[2]](#footnote-2)

### Putting It Together

Figure 5 EASY Object Model puts the previous figures together into one and adds some details such as attributes and enumeration values.

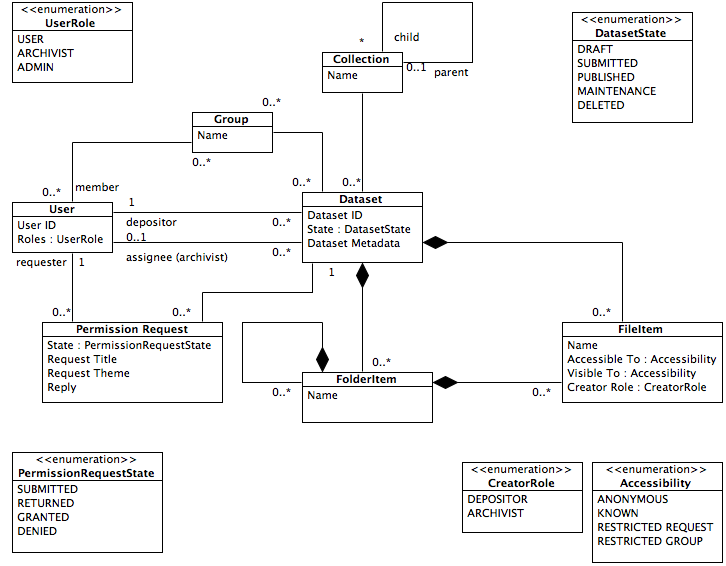


Figure 5 EASY Object Model

## Use cases

Now that we have an idea about the conceptual interface that EASY exposes to its users, we will proceed to listing the use cases that the current system supports.

### Register User Account

User requests a user account. EASY requires User to provide a minimal set of personal information and to agree to the conditions of use.

### Deposit Dataset

Known User requests a dataset to be created. EASY requires User to supply metadata and data files for the dataset. EASY also requires User to agree to deposit licence and then creates Dataset with status “submitted” recording User as depositor.

### Process Dataset

Archivist does checks on dataset with status “submitted”. Negotiates final form of dataset with depositor, then publishes dataset. Dataset becomes discoverable.

### Discover Datasets

User asks EASY for datasets matching certain metadata. EASY provides list with matching dataset IDs.

### Download Dataset

(Known) User requests download of all or selected of files of dataset. EASY checks that all files in selection are accesible to User and requires User to agree to conditions of use, then sends files to User.

### Harvest Dataset Metadata

User requests metadata records in one of available formats for all datasets in the repository. EASY returns metadata records in requested format.

### Use Case Diagram

Figure 6 Use Cases Implemented By EASY displays the use cases described above and the type of user that can access the use case. The arrows indicate specialization, e.g., a Known User is a special case of a User (specifically a User that has identified themselves by logging in). A specialization inherits all properties and behavior of its parent but may add to or modify it. In more concrete terms this means that an Archivist can do everything a Known User can, and a Known User can do everything a User can.

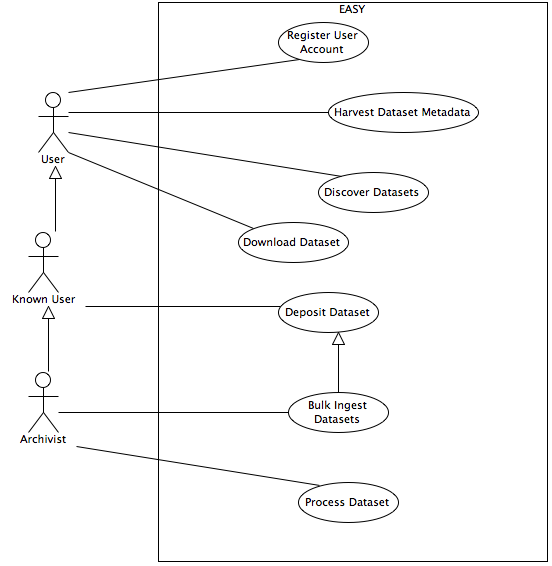


Figure 6 Use Cases Implemented By EASY

# 3 Process View

The process view describes:

1. what processes make up a running EASY system;
2. how these processes inter-communicate.

## Overview

As described in *Section* 1 Conceptual View, a complete EASY system consists of several logical components. Each of these components uses one or more processes to realize the actual behavior described in *Section* 2 Functional View*.*

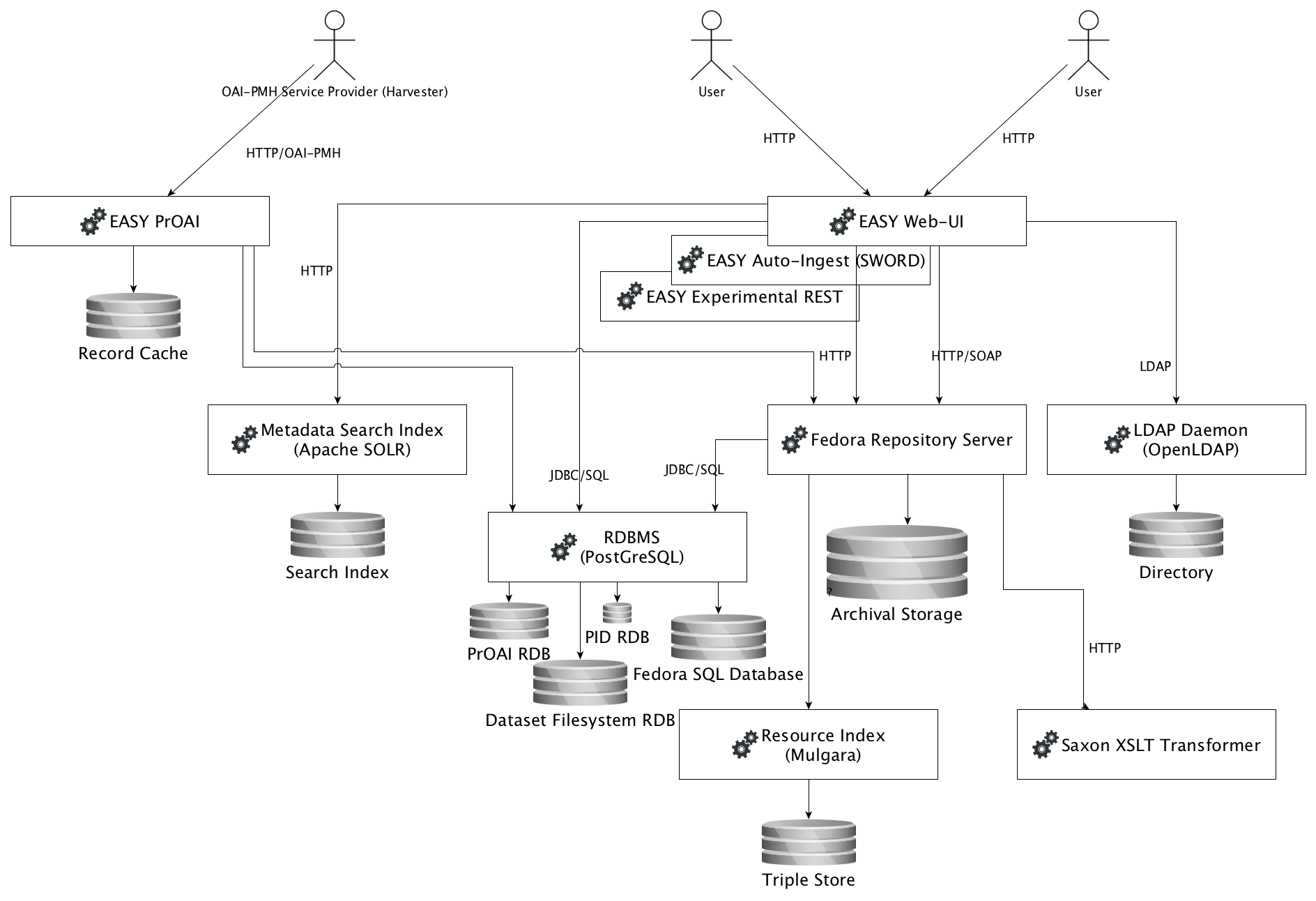


Figure 7 EASY Processes Overview

Figure 7 EASY Processes Overview shows the processes involved in a running EASY system (the boxes with the cogwheels icon), the users (stick figures) and the storage entities (cilinders). The arrows represent the communication channels. They are labeled with the protocol used in the communication. The direction of the arrow indicates which side takes the initiative to connect.

### Processes and Threads

A process may have several threads of execution. This is how a server process (such as PostGreSQL) can work on requests from several clients simultaneously.

## Processes and Protocols

We will now look at each of the processes in turn.

### EASY Web-UI, Auto-Ingest, REST

The processes EASY Web-UI, EASY Auto-Ingest (SWORD) and EASY Experimental REST all have the same structure. They all use the same libraries to access the other processes[[3]](#footnote-3). We will therefore discuss only the Web-UI, considering that the other two basically work the same way.

EASY Web-UI is a Java web application. It is the main front-end application of EASY.

The Web-UI communicates directly with the Fedora Repository Server over http. In some cases it uses Fedora’s SOAP-interface and in others Fedora’s RESTful interface. For example, if a user requests to download a data file from the repository, this results in an http request from the Web-UI to Fedora Repository Server to retrieve the file data. The Web-UI then receives the file data en sends it back to the user.

The Web-UI furthermore communicates with an RDBMS (PostGreSQL), using JDBC and SQL, to access the Dataset Filesystem RDB. As discussed in Section 1 Conceptual View, this database contains the file and folder hierarchy of the datasets. This information is also present in Archival Storage but through the Data Filesystem RDB it can be accessed much more rapidly. An example of such an interaction is when the Web-UI needs to present a tree-view of the files and folders in a dataset in its “file explorer” widget.

When the repository is searched for datasets matching certain metadata, a request is sent from the Web-UI to the Metadata Search Index (SOLR). This process keeps an index of search terms to dataset identifiers.

Finally, all user settings are kept in the LDAP Directory. When user settings need to be retrieved or updated the Web-UI communicates through the LDAP protocol with the OpenLDAP daemon.

Note that the responsibility to keep the Metadata Search Index and the Dataset Filesystem RDB up-to-date is put in the client applications discussed in this subsection. In other words, each client application must make sure that updates to the repository are correctly reflected in the Metadata Search Index and the Dataset Filesystem RDB.

### EASY PrOAI

EASY PrOAI is a special case. It does not have the same structure as the other three front-end applications. It only communicates with Fedora Repository Server to query it for changes in the digital objects stored in the repository. The queries are formulated in terms of the RDF-properties stored in the digital objects. After discovering which digital objects have changed EASY PrOAI retrieves disseminations for thoses objects. These disseminations contain metadata in several formats that EASY PrOAI needs to provide to harvesters. These metadata records are kept in EASY PrOAI’s Record Cache. For fast look-up of records in the cache EASY PrOAI uses a supporting database in the RDBMS-process.

### Fedora Repository Server

This is the gateway to Archival Storage. Fedora exposes several interfaces for searching, retrieving and managing the contents of the repository. There are two supporting services: the “Fedora SQL Database,” which supports fast look-up of digital object files in Archival Storage and the “Resource Index,” an RDF triple store that supports fast RDF-queries. For communication with the SQL Database Fedora presumably uses JDBC, and for communication with the Resource Index http.

### RDBMS (PostGreSQL)

This process handles requests for several relational databases in the system:

* The Fedora SQL Database – supports fast look-up of files maintained by Fedora in Archival Storage;
* the EASY Dataset Filesystem RDB – supports fast look-up of the parts of an EASY Dataset;
* the EASY PID RDB – a minuscule database storing the last assigned Persistent Identifier;
* the EASY PrOAI database – supports fast look-up of the location of metadata records in the Record Cache.

All the processes that communicate with the RDBMS use JDBC as a protocol.

### Resource Index (Mulgara Triple Store)

This process handles queries and updates to the “Resource Index”. Fedora is the only process that uses it directly. Other processes that want to query the Resource Index must send their requests to Fedora Repository Server, which then forwards the request to the Resource Index.

### Saxon XSLT Transformer

The Saxon XSLT Tranformer is a service that performs an XLST stylesheet transformation on an input XML. The protocol used is http-based. The Fedora Repository Server calls Saxon as part of an EASY-specific dissemination. In other words, Fedora has been configured to call Saxon when requested to return certain disseminations of a digital object. More specifically, the digital objects concerned represent EASY datasets and the disseminations are descriptions in several metadata formats such as Dublin Core, NL\_DIDL and CARARE.

### LDAP Daemon (OpenLDAP)

This is a regular LDAP service. EASY stores its user accounts in an LDAP-directory. Queries and updates are done directly to the daemon in the LDAP protocol.

### Metadata Search Index (Apache SOLR)

This process handles queries and updates to an Apache SOLR index that supports fast look-up of datasets by means of metadata queries. The protocol used is http-based.

## Example Interactions

TO DO

# 

# 4 Software View

The software view describes:

1. what software artifacts are produced and maintained at DANS;
2. the architectural rules that guide development of those artifacts.

## Overview

As described in *Section* 3 Process View, the behavior of EASY is realized through a number of processes, i.e. running instances of software programs. Most of the software programs concerned are *not* developed at DANS. Let us call those “third party” programs. An example is Fedora Repository Server, which is developed by DuraSpace. The programs that *are* developed at DANS are the subject of this section. These are:

1. EASY Web-UI
2. EASY Auto-Ingest (SWORD)
3. (EASY Experimental RESTful Service)
4. (EASY PrOAI)

The last two items are in parentheses. These are built from source code but not really being developed at DANS. The Experimental RESTful Service is, as the name implies, experimental. It has been developed at DANS, but is not currently *under development*. EASY PrOAI has not been developed at DANS, but has been forked from an open source project to fix some issues. However, this has been done under pressure of circumstances, and is considered undesirable. It is therefore also not under development at DANS.

The two remaining software programs are indeed actively being developed. We will now sketch the de facto software architecture of those programs. It will be no more than some basic guidelines, as the architecture has become somewhat involved and confused over time. Also, these guidelines are not followed everywhere in the code.

## Programs

What follows some brief remarks about the software architecture of each program.

### EASY Web-UI

EASY Web-UI is a Java EE web application. Figure 8 EASY Web-UI Software Architecture, show the basic architecture.

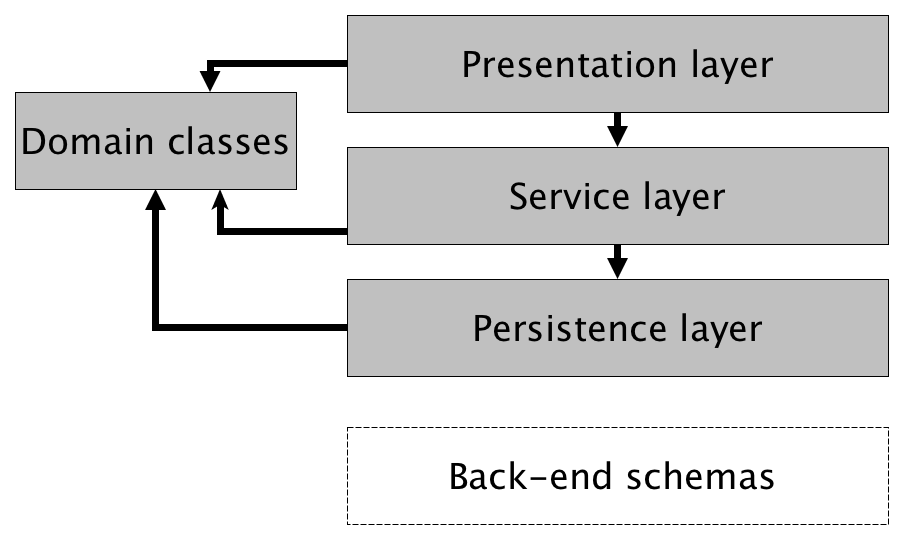


Figure 8 EASY Web-UI Software Architecture

The source code is partitioned into layers. Each layer only calls the layer beneath it. Apart from the layers there are domain classes, that define what entities there are in the application domain. The domain classes are used by all the layers and serve as a “language” to talk to each other.

There are three layers:

1. **Presentation layer** – contains the code that deals with presenting a user interface to the user. It has been implemented with the Apache Wicket library.
2. **Service layer** – exposes services to the presentation layer. The services are defined in terms of the abstract concepts that a user works with such as Datasets, File Items and User (accounts), see Section 2 Functional View.
3. **Persistence layer** – implements the lower level details of persisting the entities, for instance by storing a Dataset as interconnected group of digital objects in Fedora (by making calls to the Fedora Repository Server through SOAP).

The persistence layer is complemented by the “back-end schemas”. This is the code that creates the correct initial digital objects in Archival Storage, database schemas int the RDBMS, LDAP schemas in the LDAP directory, etc. This code could also be classified as part of the persistence layer, but it does not actually run in the layered application. Rather, it sets up things in the persistent storage locations so that the program encounters the environment it expects at run-time.

The code for the three layers is packaged together into a Java EE web archive (a WAR-file). The services and persistence logic can therefore only be reused by another application by incorporating all the same (compiled) libraries, as we shall see next.

### EASY Auto-Ingest (SWORD)

The Auto-Ingest service is also a Java EE web application. It is a machine-to-machine interface that implements the SWORD/APP protocol. The software architecture of this program is essentially the same as that of EASY Web-UI, except that the presentation layer has been replaced by an implementation of the SWORD/APP protocol.

Figure 9 EASY SWORD Software Architecture shows what is meant by the previous paragraph.

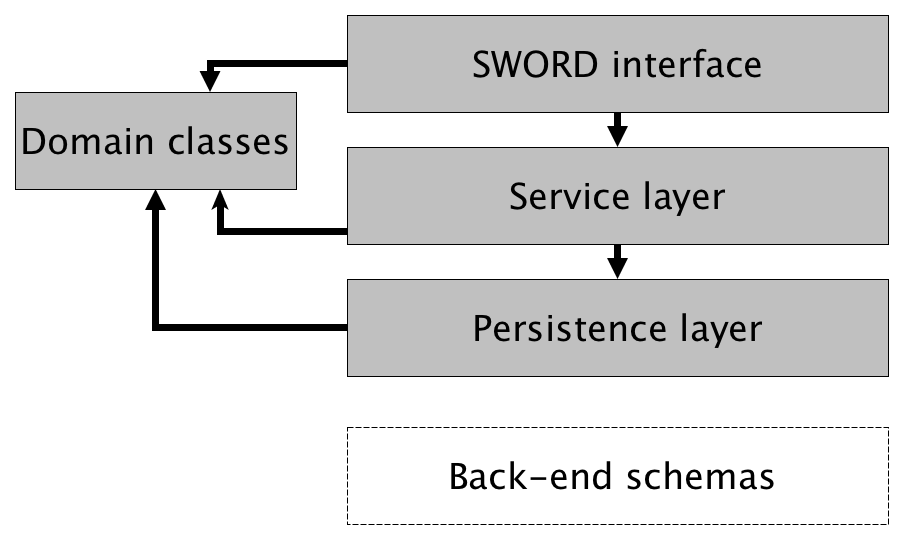


Figure 9 EASY SWORD Software Architecture

### EASY Experimental RESTful Service

At the risk of becoming monotonous, the EASY Experimental RESTful Service is also a Java EE web application and follows the same pattern as EASY Web-UI and EASY SWORD Auto-Ingest Service, in this case replacing the presentation layer with a RESTful interface to query and retrieve the EASY metadata. We will leave drawing up the diagram as an exercise for the reader.

### EASY PrOAI

The EASY PrOAI OAI-PMH Data Provider is, as mentioned previously, a forked open source project. Some performance issues in it have been resolved. At this point no decision has been made about whether to continue using PrOAI or move to some other OAI-PMH Data Provider software. If it is decided to further develop the current forked project, the software architecture should of course first be reverse engineered and properly described here.

# 

# 5 Digital Preservation View

The digital preservation view describes:

1. the OAIS information model;
2. the Fedora Commons Digital Object Model;
3. how EASY (partly) implements long term preservation using Fedora.

## Overview

Ultimately the goal of EASY is to preserve digital scientific research data for the long term. As we shall see the word “data” should actually be replaced by the word “information”, as the understandability of the data is a key part of what is meant by “preservation”.

OAIS information model

The Reference Model for an Open Archival Information System (OAIS) describes an information model that is geared towards the problems of digital preservation. We will briefly review that model to use it as a reference in the description of digital preservation support in EASY. A comprehensive description of this model can be found in the OAIS specifications and is highly recommended reading.

Fedora Digital Object Model

The Archival Storage component of EASY is organized and maintained by Fedora Commons Repository software[[4]](#footnote-4).To this end it uses a model called the Fedora Digital Object Model and an accompanying serialization format. We will briefly describe both.

EASY Object Model

EASY implements digital preservation using the Fedora Digital Object Model. We will therefore relate the way that EASY stores its holdings in Archival Storage (using Fedora) to the concepts described in OAIS. This will also reveal where EASY is lacking with respect to its long term responsibilities.

## OAIS Information Model

### Information Objects

The OAIS reference model views as the main goal of any open archival information system the preservation of information. The definition of “information” thus plays an important role.

information vs data

OAIS distinguishes “information” from “data”. Information is exchangeable knowledge. A Data Object is the encoding of a piece of information. A Digital Object is a Data Object that encodes information in digital form, that is to say in a set of bit sequences. This is in fact the only type of Data Object that a digital archive such as EASY is concerned with, so when we talk of “Data Objects” we mean “Digital Objects”.

Data Object + RI = Information Object

To turn a Data Object into an Information Object, so-called Representation Information (RI) is needed. This is information that details the format and meaning of the bit sequences that makes up the data. Examples are file format definitions and software that renders data in a visual form. The data and Representation Information together are called an Information Object. An archive should either store the necessary Representation Information or reference it in another archive.

Representation Network

As Representation Information is itself information it can be viewed as a set of Information Objects, so a “Data Object + RI”. This gives rise to a network of Information Objects called a Representation Network. An archive should be explicit about where the nodes of this network are located.

transparency to the bit level

One important thing to note about Information Objects is that they are different from objects in the sense of object oriented programming in that there is no “information hiding”. Quite to the contrary, “information exposure” is the central goal. This is what OAIS calls the “requirement of transparency to the bit level”.

Designated Community and Knowledge Base

A qualification is in order. Of course, the question arises how much Representation Information is sufficient for data to be turned into information. This all depends on the “readers” of the data and their previous knowledge. In OAIS this is captured in the concepts of the Designated Community and its Knowledge Base. These should be well-defined in order for a preservation strategy to be evaluable.

### Information Packages

AIP: CI and PDI

In OAIS, Information Objects are the building blocks of archival holdings. The things stored in an OAIS archive are packages of information objects, aptly called Archival Information Packages (AIP). As well as the target of preservation (the Content Information, or CI) an AIP contains Information Objects that support the preservation function, such as checksums, identifiers, context information and provenance. These objects together are called Preservation Description Information (PDI).

packaging information

AIPs are identified and bound together by Packaging Information. In practical terms this means that the Packaging Information allows the archive to collect all the Information Object needed to reconstruct the AIP.

SIP and DIP

The packages ingested into the archive and disseminated from it are also Information Packages. They are called Submission Information Packages (SIP) and Dissemination Information Packages (DIP), respectively. They have the same components as AIPs, but these components are now optional.

AIC and AIU

AIPs can contain other AIPs. AIPs that do are called Archival Information Collections (AICs). AIPs that do not, are called Archival Information Units (AIUs).

## Fedora Commons Digital Object Model

### Fedora Digital Objects

Fedora Commons is designed to be a repository that supports long term preservation. It is built around a simple digital object model with an XML serialization format (called FOXML) that is intended to be preservable.[[5]](#footnote-5)

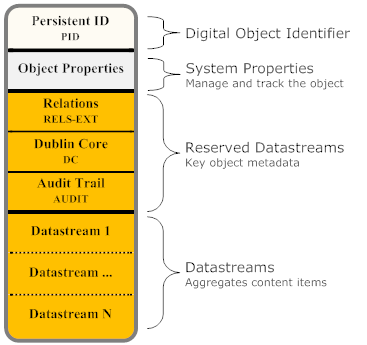


Figure 10 Fedora Commons Digital Object Model

Figure 10 Fedora Commons Digital Object Model (taken from the Fedora Commons website) shows the components of a Fedora Digital Object (FDO)[[6]](#footnote-6):

* an identifier, unique within the repository;
* system properties, such as modified date and owner, maintained by the Fedora Respository Server;
* reserved datastreams for (also maintained by the Fedora Repository Server, although it provides an API for making changes to them):
  + recording relations to other FDO’s in RDF (RELS-EXT);
  + recording basic metadata about the FDO (DC);
  + recording changes to the FDO (AUDIT);
* datastreams, for the user to store content in.

The serialization format FOXML directly reflects the abstract structure outlined above. See Figure 11 FOXML Structure.

<digitalObject PID="uniqueID">

  <!-- there are a set of core object properties -->

  <objectProperties>

    <property/>

    <property/>

    ...

  </objectProperties>

  <!-- there can be zero or more datastreams -->

  <datastream>

    <datastreamVersion/>

    <datastreamVersion/>

    ...

  </datastream>

</digitalObject>

Figure 11 FOXML Structure

storage of datastreams

Where a datastream is stored depends on its “control group”. There are four possibilities:

* Inline XML – stored in the datastreamVersion element in FOXML. This is only possible if the datastream consists itself of XML;
* Managed Content – stored in a location controlled by Fedora;
* Redirect – stored externally, only a URL to the location is kept by Fedora to *redirect* clients to it;
* External – stored externally, only a URL to the location is kept by Fedora to *retrieve the content* when requested.

The exact way that Fedora stores its holdings is configurable. The storage logic is hidden behind the Fedora Low Level Storage interface. The default implementation uses the filesystem, but there are also implementations that use S3- or blob-storage.

## Digital Preservation in EASY

We are now ready to describe how EASY uses Fedora to implement its digital preservation task.

### EASY Object Model in Fedora Digital Object Model

In Section 2 Functional View we described the object model that EASY presents to its users. The model hides many implementation details from the user. As we saw above (see Sub-section Information Objects) OAIS explains why it is necessary to expose the archive down to the bit level.

The difference in approach is due to the different interests of the viewers. The “Functional View” takes the point of view of a normal user of the system, for example a researcher who deposits data or is searching for data in the repository. Such a user is quite happy to take for granted that EASY stores its data in a preservable way. However in this view, the “Digital Preservation View”, we are taking the more critical stance, let us say, of a funding agency or a digital archive certification organization, such as DSA or DIN. We therefore need to examine how things are stored exactly, down to the bit level so as to prove that no information will be lost.

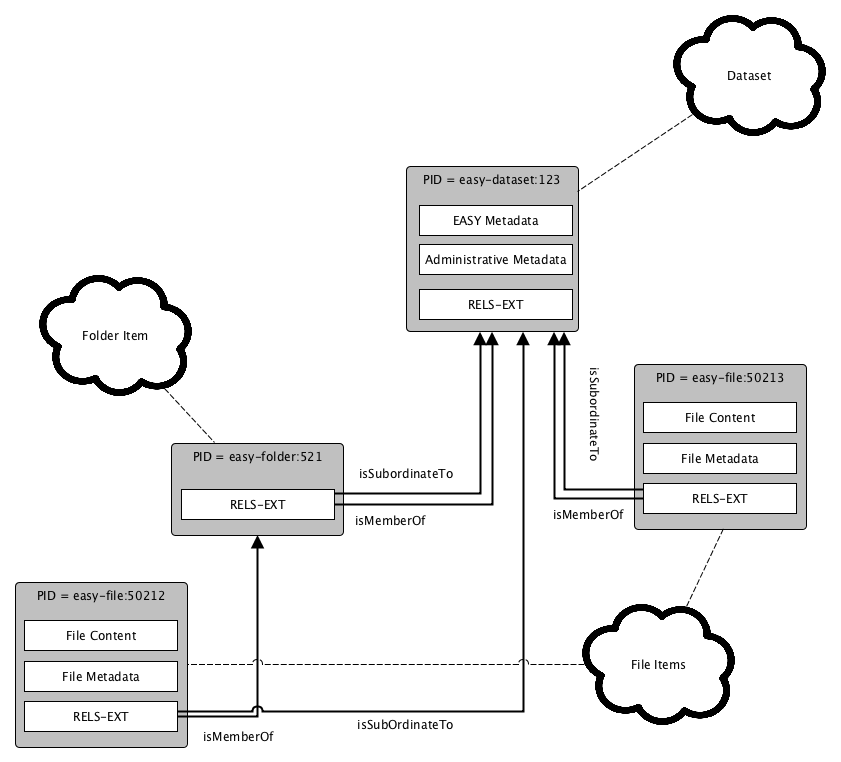


Figure 12 EASY Object Model in Fedora Digital Objects

Figure 12 EASY Object Model in Fedora Digital Objects shows an example of how the EASY Object Model is implemented in the Fedora Commons Repository. This example shows an EASY Dataset with two files in it, one of which is stored in a folder. The RDF-predicate *isMemberOf* records that an item (File Item or Folder Item) is a direct child of a container (Dataset or Folder Item). The predicate *isSubordinateTo* indicates that an item is part of some Dataset.[[7]](#footnote-7)

Note that the arrows indicating these relations start from the RELS-EXT datastream. This is done to highlight the fact that the corresponding RDF-statements are recorded in this datastream.

### OAIS Analysis of EASY Datasets

Let us now analyze the EASY Object Model implementation in terms of the OAIS information model.

Since OAIS is a reference model it leaves some room for interpretation. Specifically, the mapping of some OAIS concepts to implementation entities depend on choices made by the archive. These choices have not always been made by DANS, or at least not explicitly. This means that the following description may not be the only possible one. Of course, the attempt has been made to make sensible mappings.

#### AIPs

AIPs

The first question is: what is an AIP in EASY? The most obvious answer is: a Dataset is an AIP. Furthermore, the hierarchical structure of Datasets leads us to suspect that AIPs in EASY are also hierarchical, i.e. Datasets are AICs with zero or more AIPs as Content Information, each of them again an AIC (Folder Item) or AIU (File Item).

|  |  |
| --- | --- |
| OAIS | EASY |
| AIP – AIC | Dataset |
| AIP – AIC | Folder Item |
| AIP – AIU | File Item |

Content Information

The Content Information of an AIC consists of everything that is in the FDO’s that are hierarchically under it (members, member of members, etc). So, for example, the Content Information of the Dataset “easy-dataset:123” consists of the FDO’s “easy-folder:521”, “easy-file:50212” and easy-file:50213”. Also, the Content Information of Folder Item “easy-folder:521” consists of only the File Item “easy-file:50212”. The Content Information of an AIU clearly is *at least* the EASY\_FILE datastream, which contains the Data File stored in the File Item. However, it is not entirely clear where the Representation Information is located. It is probably not complete or absent in most cases. The MIME-type, which is stored in the File Item Metadata could be viewed as a very weak form of structural Representation Information. In some cases information about the semantics of a file may be located in another file in the same dataset. In that case the file containing the explanations would be part of the Representation Information of the Content Information. These relations are however not expressed formally, or – as far as I know – even informally.

#### PDI

The Preservation Description Informantion

|  |  |  |
| --- | --- | --- |
| OAIS Concept | Data-stream | Element(s) |
| Reference Information | EMD | emd:identifier |
| Context Information | EMD | emd:description |
|  | RELS-EXT | Records membership of disciplines and other collections (is this context information?) |
| Provenance Information | EMD | emd:creator  emd:contributor |
|  | AMD | published-> maintenance->published indicates a change to the AIP.  damd:workflowData -> record of intake by data manager |
| Fixity Information | n/a | n/a |
| Access Rights Information | EMD | emd:rights/dct:accessRights  emd:date/eas:available |
|  | DATASET\_LICENCE | complete content (pdf) |

PDI – EASY Dataset

TO DO:

* PDI
* SIPs
* DIPs

# References

* OAIS
* UML

1. This implies, by the way, that the contained element is linked to one and only one container. [↑](#footnote-ref-1)
2. Currently there is only one group that datasets can be assigned to: Archaeology [↑](#footnote-ref-2)
3. This is actually a weakness in the current architecture since, because of this, responsibilities are duplicated over several applications. [↑](#footnote-ref-3)
4. The current release version of Fedora Commons is 3.7. We are using 3.5. Version 4.0 is currently in alpha-testing phase and is based on a more complex object model. [↑](#footnote-ref-4)
5. There are two other serialization formats: Fedora Atom and Fedora METS, wich are, however, not currently used by EASY. [↑](#footnote-ref-5)
6. A Fedora Digital Object is not exactly the same as an OAIS Digital Object. To avoid confusion we will refer to it by the mnemonic FDO. [↑](#footnote-ref-6)
7. This predicate might actually be inferred from *isMemberOf* relations, however it is recorded all the same for each instance. [↑](#footnote-ref-7)