SEVENTH FRAMEWORK PROGRAMME
Challenge 1
Information and Communication Technologies

Trusted Architecture for Securely Shared Services

Document type: Deliverable

Title:	*Software Documentation* *System: Repository services*

Work Package: 8

Deliverable Number: 1

Editor: Marc Santos, University of Koblenz-Landau

Dissemination Level: Public

Preparation Date: 1 June 2009

Version: 1

The TAS3 Consortium

Nr	Participant name	Country	Participant short name	Participant role
1	K.U. Leuven	BE	KUL	Coordinator
2	Synergetics nv/sa	BE	SYN	Project Manager
3	University of Kent	UK	KENT	Partner
4	University of Karlsruhe	DE	KARL	Partner
5	Technische Universiteit Eindhoven	NL	TUE	Partner
6	CNR/ISTI	IT	CNR	Partner
7	University of Koblenz-Landau	DE	UNIKOLD	Partner
8	Vrije Universiteit Brussel	BE	VUB	Partner
9	University of Zaragoza	ES	UNIZAR	Partner
10	University of Nottingham	UK	NOT	Partner
11	SAP research	DE	SAP	Partner
12	Eifel	FR	EIF	Partner
13	Intalio	FR	INT	Partner
14	Risaris	IR	RIS	Partner
15	Kenteq	BE	KETQ	Partner
16	Oracle	UK	ORACLE	Partner
17	Custodix	BE	CUS	Partner
18	Medisoft	NL	MEDI	Partner

Contributors

	Name	**Organisation**
1	John Power	Risaris
2	David Chadwick	University of Kent
3		
4		
5		

Table of Contents

Table of Figures

Executive Summary

This document provides the description of '*Repository Services'* – a component of the TAS3 Trusted Application Infrastructure.
Repositories are needed in TAS3 to store personal identifying information. Moreover, they need to be able to also store so called 'Sticky Policies' with those data items.
The TAS3 Trusted Application Infrastructure is the application dependent part of the TAS3 infrastructure. Its purpose is to provide the services needed to realize the pilots of WP9 in the fields of employability and eHealth. The TAS3 Trusted Application Infrastructure depends on the requirements collected in WP1, the architecture design provided by WP2 and the business process models developed in WP3. It allows the application independent services developed in WP4, WP5 and WP7 to be used in the WP9 pilots. Most of the concrete technical solutions in this deliverable are inherited from deliverable 4.2. [1], which has a more generic view on Repositories.

Within the TAS3 Trusted Application Infrastructure the 'Repository Services' component serves to store, protect and deliver PII[1] (Asset Banks). This document describes

- The Conceptual model - the assumptions on which the development of 'Repository Services' is based.
- An evaluation of existing Open Source repositories and a description for a chosen TAS3 reference repository.
- The services provided by 'Repository Services' and a description of its architecture.
- Plans for the further development of the services in the next phase of the TAS3 project.

NOTE: The software components produced in WP 8 implement application specific adaptors that are required to use the application independent TAS3 infrastructure in the TAS3 pilots in eHealth and eEmployability. The overall architecture, semantically enriched executable business process models with an XForms user interface and the design of the core TAS3 services are the pre-requisites for WP8. The TAS3 architecture has been finalized only recently (see D2.1). While all our results are consistent with and usable for the current TAS3 architecture it cannot be a surprise that some alignments and refinements will be required. Moreover new service needs, for example the request for a service bus, are emerging from the architecture document, which still need to be detailed before they can be implemented and documented in any of the deliverables of WP8.

Technical Note:
All produced components (web services, libraries and clients) of Deliverables D8.1., D8.2. and D8.3. can be found in a binary version at this location:
http://citrix.uni-koblenz.de:9000/homepage/tas3/default.aspx

To access this page, please use the following login data:
Login: tas3
Password: z65rf5

Readers Guide

To help the reader of this document to better understand its contents together with the other two deliverables of WP8 (especially Deliverable D8.2.), we provide this readers guide.

Deliverables D8.2. is based on this deliverable D8.1.

---

[1] **personal identifying information**

D8.1. has many references to D8.2., because D8.2. describes services that extend the repository services described in this document.
Section 2 in D8.1. is the section that contains the conceptual model for TAS3 repositories. This conceptual model is then filled with flesh by section 4 and 5 from this document and section 2 in deliverable D8.2.
The reader can use section 2 in D8.1. as a starting point for reading deliverable D8.1. and D8.2.
Section 6 (applied software engineering methods) in this deliverable is optional and in an initial state, but we decided to leave it in the document, because this document is a software documentation and we have to document our applied software engineering methods.

1 Introduction

1.1 Scope, Objectives and Context

This deliverable describes the implementation of the so called 'ADPEP' (ADPEP stands for **A**pplication **D**ependent **P**olicy **E**nforcement **P**oint) for a TAS3 repository. The ADPEP is a major component in the TAS3 application dependent infrastructure. It is responsible for accessing either TAS3 databases for PII (in this case the Fedora repository) or legacy databases, which is done over the TAS3 adapted Risaris SOA gateway.

This ADPEP for a TAS3 repository consist of two parts: A 'ServiceResponder' for ingoing request and a 'ServiceRequester', which allows a repository to initiate requests. The focus of implementation work in this first phase of the project was on the 'ServiceResponder' part, because this functionality *must* be provided by a TAS3 repository. The initiation of requests by a repository is optional.

For a better understanding the following figure tries to illustrate the function and the position of the 'ServiceResponder ADPEP' in the TAS3 infrastructure (see right box –a more detailed description is available under section 4).

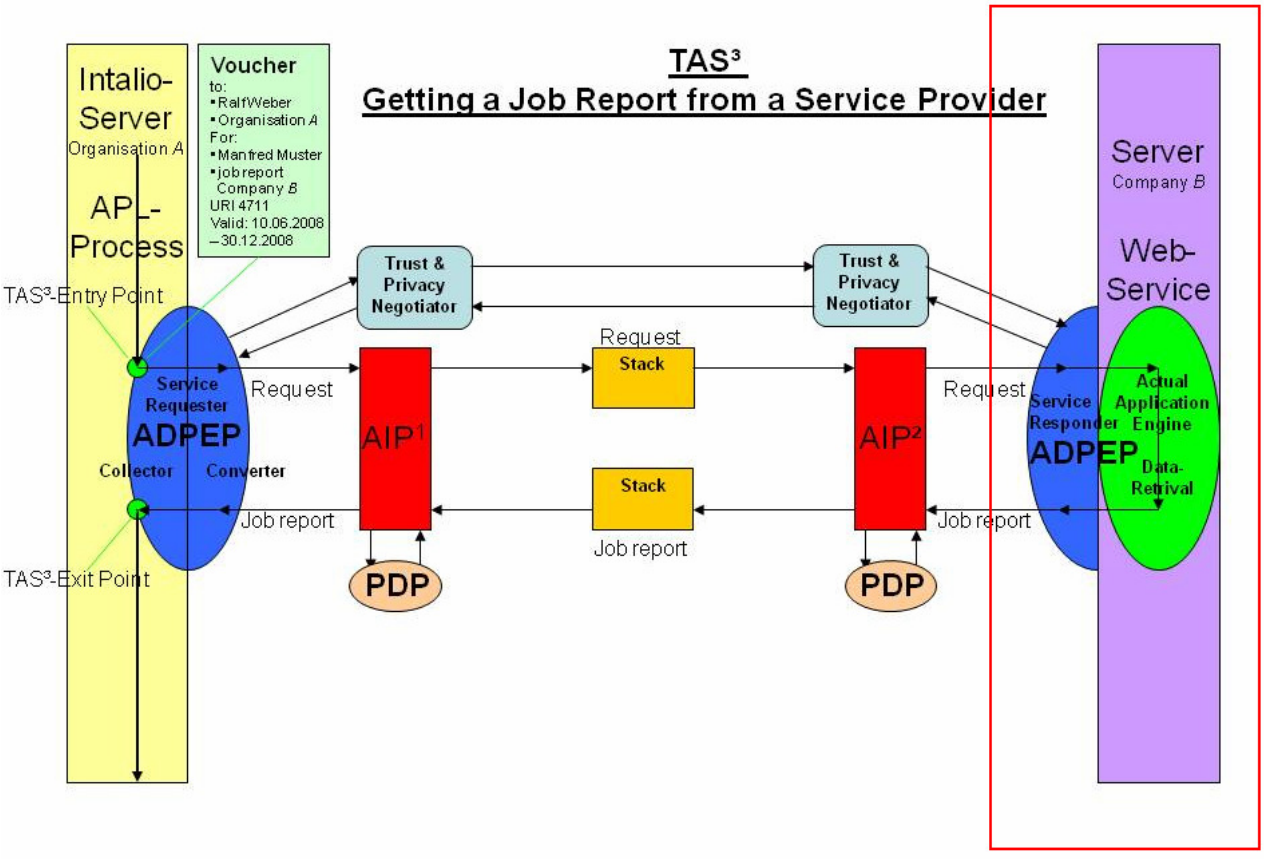

Figure 1: Simplified conceptual overview over the TAS3 infrastructure.

The 'ServiceResponder ADPEP' is part of the 'Service Provider' (the red surrounded box contains the Service Provider with the 'ServiceResponder ADPEP' and an 'Application Engine: e.g. a repository).

The 'ServiceResponder ADPEP' is implemented and accessible as a web service over SOAP[2]. Moreover the chosen repository for TAS3, which is the Open Source repository 'Fedora' (more details in section 3), also provides web service interfaces. That makes it simple to integrate those two components into the 'Service Oriented Architecture' of TAS3.

Even though WP8 is not directly responsible for security related components according to the description of work 'This work package will implement all components that are not directly security- or trust related and that are needed to create working systems that can cover the envisaged use cases' (see DoW chapter **B.1.3.5.8**), we have to integrate those security related components and to provide interfaces for attaching security information to messages. Each major component in WP8 has always a security aspect and a special interface to support the security and trust components from WPs 4 until 7.

This document focuses on the description of the components on the Service Provider side (the right side in the upper figure). More details on the Service Requester (the left side) can be found in deliverable D8.3. [10]. More details on the complete architecture can be found in deliverable D2.1. [2]

[2] http://www.w3.org/TR/soap/

1.2 Structure of this Document

Section 2 describes the 'Conceptual Model' for TAS3 secured repositories. It is the starting point for the rest of this document and could even be used as starting point to read Deliverable D8.2., which is about 'Auxiliary or Back Office Services'.

Section 3 is about the chosen Open Source repository that will be extended by TAS3 services. In the Annex is more detailed information with further explanations of the chosen reference repository: *Fedora*[3].

Sections 4 and 5 fill the conceptual model for TAS3 repositories with flesh. Section 4 is about accessing, managing and storing personal identifying information (PII) in TAS3 by a special TAS3 secured repository.
The Fedora repository is not usable for TAS3 requirements without any modifications or extensions. The first extension we can provide is called the 'TAS3FedoraHelperLibrary'. This library is encapsulated as a web service and can be stacked on top of the Fedora system. It is the mediator service between the TAS3 core infrastructure (security and trust layer) and the service provider or -more specifically- the Data Provider. The TAS3FedoraHelperLibrary is the main component in the so called 'ServiceResponder ADPEP' web service.
This web Service, its underlying main library and a desktop client for testing purposes are introduced in section 4. Moreover, the security and trust related interfaces in the ServiceResponder ADPEP are introduced.

Section 5 is about the same issue but without the TAS3 secured repository. Instead TAS3 tries to integrate legacy databases using the RISARIS SOA gateway (also secured by TAS3).

In the 6th section we explained some common guidelines of our implementation concerning unit testing and security aware programming.

Gender generalization: Wherever the male form (he/him/his) is specified the female is also implied!

[3] http://www.fedora.info

2 Conceptual Model

2.1 Introduction

The purpose of this section is to describe the conceptual model for TAS3 conformant repositories. This section is concerned with both the security and non-security related functions of data repositories, but primarily with the later since the security concerns are addressed in much more detail in Deliverable D4.2 [1] and D7.1 [5]. Figures 2 to 5 show how the functional components are used for processing both incoming and outgoing messages, for both the Encapsulated Security Protocol (ESL) approach and the Application Protocol Enhancement (APE) approach.

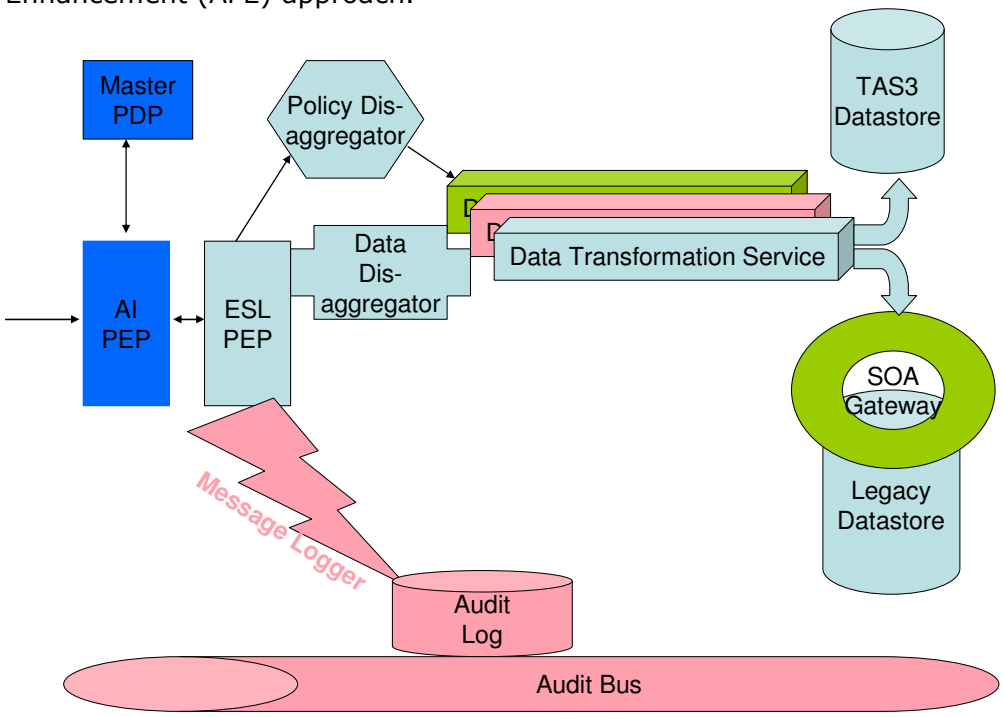

Figure 2: Incoming Message using the ESL protocol

Figure 2 shows an incoming message to a TAS3 repository, via the Application Independent PEP (AIPEP). The message could be a new request from a client, or a response to an outgoing request from the TAS3 repository e.g. to import some data from another TAS3 repository. In this scenario, the AIPEP talks the (still to be defined) TAS3 Application Layer protocol to the other party. The ESL PEP is passed the application layer data, the sticky policy and the incoming message from the AIPEP. The PEPs are described in more detail in Section 2.2 below. The first task of the ESL PEP is to log the message, using the logger which is briefly described in Section 2.5. The PEP may need to dis-aggregated the application layer data and sticky policy and this is briefly described in Section 2.4. The application layer data may also need to be transformed before storage, and the data transformation service is briefly described in Section 2.3. Finally the optionally disaggregated and transformed data can be stored in the TAS3 data store. If the data store is TAS3 conformant then it can be stored "as is". If the data store is a legacy database, then the interface to it is the SOA gateway. This is described briefly in Section 2.6 and in more detail in Section 5.

Figure 3 shows an outgoing message from either a legacy data store or a TAS3 repository. The message may be an outgoing response to a request for information, or it may be a request initiated by the repository. Any data in the message may initially need to be aggregated (see Section 2.4), then transformed (see Section 2.3), coupled with its sticky

policy (which may also need to aggregated from multiple policies), and finally the message is logged (see Section 2.5) before being passed to the AIPEP for transfer to the other party.

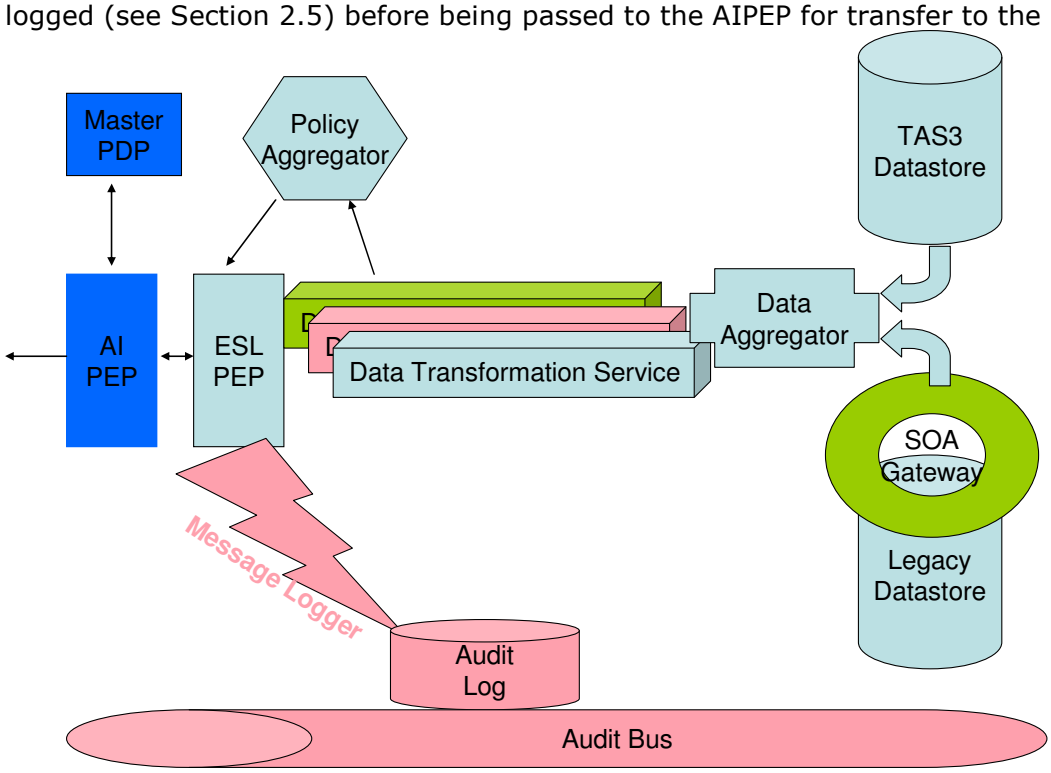

Figure 3: Outgoing Message using the ESL protocol

Figures 4 and 5 show the same scenario, only in this case the incoming message has been directly received by the PEP via the application layer protocol enhanced (APE) so that it can carry the sticky policy as well as the application layer data.

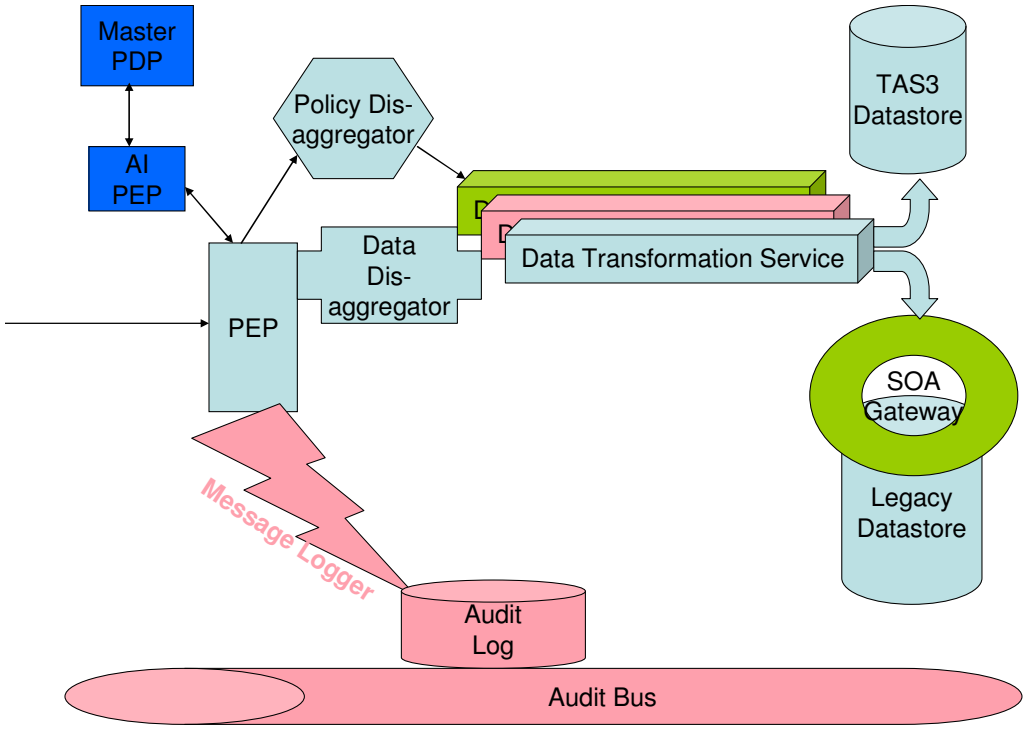

Figure 4: Incoming Message using the enhanced application layer protocol

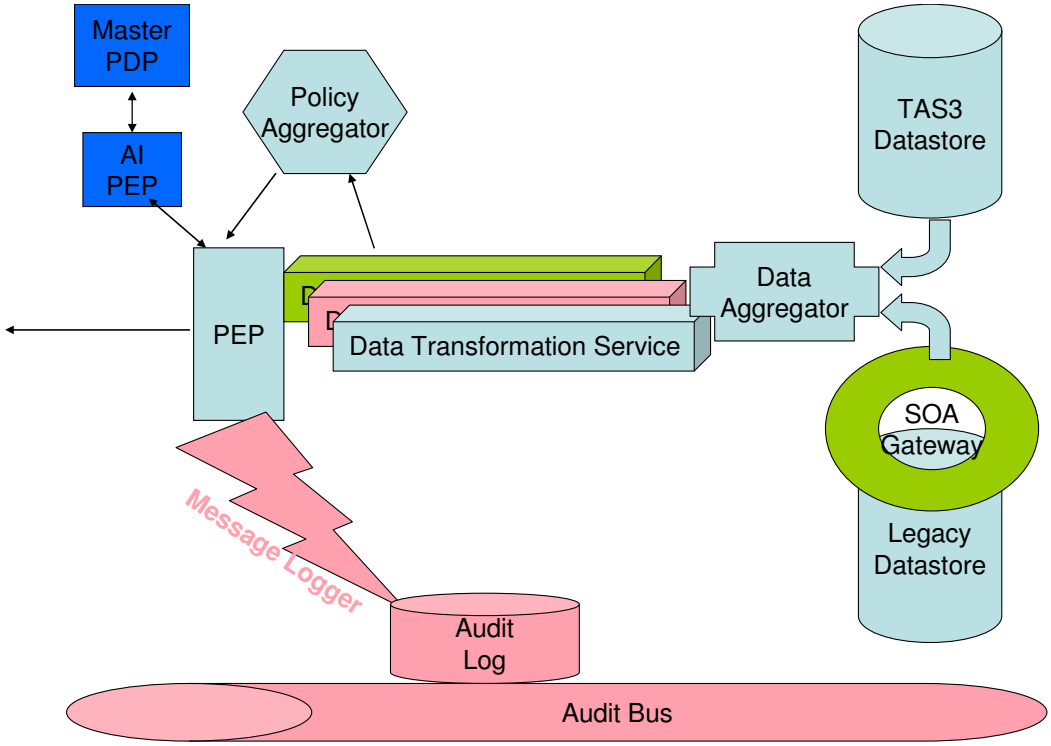

Figure 5: Outgoing Message using the enhanced application layer protocol

2.2 The PEP front end

Two different repository PEPs will need to be constructed depending upon whether the ESL or APE protocols are being used.

The ESL PEP will communicate with the remote party via the TAS3 Application Layer protocol which is still to be defined (by WP7), and with the AIPEP via a TAS3 standard protocol interface, which is also still to be defined. These will be documented in the next version of this deliverable and in deliverables of WP7. The initial prototype implementation described in Section 4 uses Serialised Java Objects as the Application Layer protocol, but this was simply an expedient measure to expedite testing of the first prototype.

The APE PEP will communicate with the remote party via the existing application layer protocol, which will be enhanced to support the carrying of sticky policies. For example if the TAS3 repository uses the LDAP protocol [9] to store and retrieve data, then the APE approach could define an LDAPv3 protocol *control* in which to place the sticky policy, or could define an LDAP attribute in which to hold it, or both. If the repository uses SQL commands for accessing the data, then the APE approach could either enhance the proprietary SQL protocol defined by the particular database vendor in order to carry the sticky policy, or could define a new SQL table to hold the sticky policies and relate these to the data they control. The APE PEP will communicate with the AIPEP via a TAS3 standard protocol interface, which is still to be finalised, but is most likely to be the SAML 2.0 Profile of XACML [8] or a slight variant of it. This will be documented in the next version of this deliverable, along with at least one enhanced existing application layer protocol.

Obviously an ESL PEP and an APE PEP will not be able to directly communicate with each other since they talk different application layer protocols. A TAS3 trust network may therefore decide to adopt one approach or the other. However, a trust network can support both approaches simultaneously by using a discovery service to inform requestors which

approach a particular TAS3 repository uses. A repository can use both approaches and offer both protocols via different service endpoints.

2.3 Data Transformation Service

Data transformation is an optional service of a repository. Repositories are always capable of storing whatever data the user provides and returning it "as is", by treating the data as a binary blob and using a null transformation service. But this often limits the usefulness of the data in a repository. The purpose of the data transformation service is to allow the re-use of repository data, in particular in cross-domain settings. Different domains have different technical standards and specifications for their data, for example, HR-XML[4] for the employability sector and IMS LIP[5] for the educational sector. In order to use the same PII in these different sectors, data transformation is needed. If TAS3 were to provide transformations between any two relevant formats this would soon become unmanageable. It is not a scalable solution. The only feasible approach is to use a single core *generic* data format and to provide translations to/from each relevant format and this core data format. This is the purpose of the Generic Data Format described in Deliverable D8.2 [6] Section 2.1.2.

Transformation services are necessary to meet the requirement of cross domain data re-use and so the usual approach will be for users to request their data be converted into the generic data format during storage, and then transformed into the required format on retrieval. Data transformation is described more fully in D8.2.

2.4 Data and Policy Aggregation/De-aggregation Services

The purpose of the data aggregation service is to create a composite data object (or collection) from smaller data objects, for example, the creation of a Personal Details collection from Name, Address, Date of Birth, and Marital Status data objects. Collections are specified using declarative rules. Dis-aggregation is the splitting of a collection into its constituent data objects. Both processes can be applied recursively, for example, a Curriculum Vitae could be composed from Personal Details and Employment Details data objects, and an Address object from Street Name, City, Country and PostCode objects. Data aggregation is specified in more detail in Section 2.2 of D8.2.

The policy aggregation service is used to aggregate the sticky policies of data objects that have been aggregated into a collection as described above. Similarly the policy de-aggregation service is needed to extract the individual sticky policies that need to be stuck to dis-aggregated data objects. Policy aggregation and dis-aggregation is described more fully in Section 2.2.9 and Annex 4 of D8.2

2.5 Message Logger Service

The purpose of the message logger service is to extract the necessary logging information from incoming and outgoing messages and to send:

i) the complete set of logging information to a secure audit log
ii) a summary of the logged information to the Audit Bus

The logger service needs to be configurable so that the repository security administrator can determine what information should be logged (both complete and summary information). Furthermore the message logger should provide a service interface to allow the logger to be dynamically adapted to fit the current security context. For example, if an intrusion detection system suspects that the repository is being attacked then the logging might be switched to

[4] http://www.hr-xml.org
[5] http://www.imsglobal.org/profiles/

High level, whilst under normal operations it might be switched to Low level. The initial implementation of the logger service is described in D8.2.

2.6 Legacy Systems + the SOA Gateway

Figure 6 shows a more accurate representation of the functions of the SOA Gateway. In essence the SOA Gateway provides all the TAS3 security and non-security functions that are needed to convert a legacy datastore into a TAS3 repository. The SOA Gateway will directly communicate with the remote party using the enhanced application layer protocol (as in Figure 6) or via the AIPEP if the ESL approach is being used (not shown). The SOA gateway will communicate with the legacy datastore using the datastore's existing protocol. This means that the SOA gateway has to be tailored to communicate with each type of legacy datastore that it supports. Section 5 describes one implementation of an SOA Gateway that is being provided by Risaris.

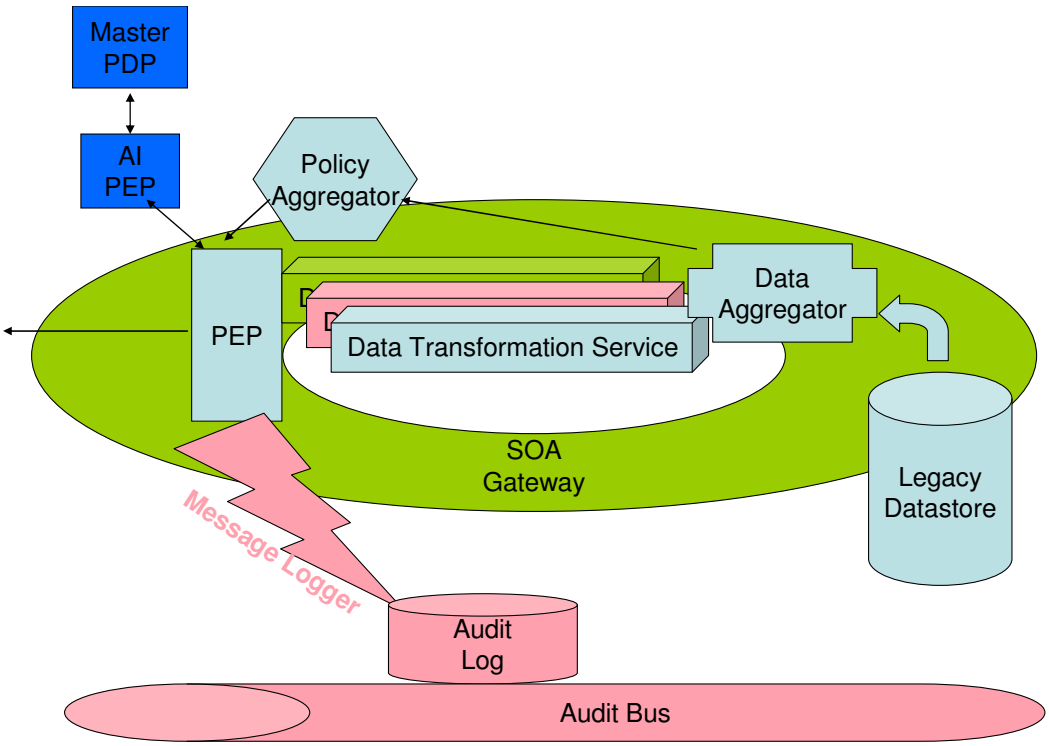

Figure 6: The SOA Gateway

3 Choice of TAS3 Reference Repository System

In this chapter, we describe the decision making for a reference repository used for the demonstrators in TAS3. Then we describe the available repository functionalities and its interface.

This decision is based on contents from

- Chapter 2, where we explain the conceptual model of TAS3 repositories.
- And Deliverable 4.2[1], which describes the functionality of a **secure** repository in TAS3. In Deliverable 4.2 the reader can find a short definition of a repository and how TAS3 understands the role of it in TAS3.

3.1 Requirements for a reference repository in TAS3

3.1.1 Repository definition

Definition for repositories in TAS3 (deliverable 4.2):

'***Repositories are web enabled containers for storing digital information***'

In TAS3 this digital information is always person related or one may call it **personal identifying information** (PII). The end user shall have an opportunity to store, modify and read his own data in a simple way. Moreover he needs to give others access to the data objects. Those requirements imply that the repository has a mechanism to store digital objects in a secure way and an authorization service that allows 'sticky' access control in a fine grained way.

3.1.2 The repository is part of the Service Oriented Architecture

The repository needs to be **web enabled** and has to **provide its services over web services (may they be based on SOAP[6] or REST[7])**.

According to the architecture (described in deliverable 2.1 [2]) secure repositories have web services interfaces allowing them to be accessed directly by the TAS3 infrastructure. In our initial implementation we have chosen instead to place a security gateway in front of a partially secure repository that catches all incoming requests. This security gateway is also described in this deliverable: it is called the ServiceResponder ADPEP (see section 4). But there are exceptions to this architectural design. Because direct user interfaces have to be provided by the repository these will bypass the ADPEP. Instead they will use the PEP and PDP provided by the repository itself.

A user shall be able to authenticate and authorize to the repository using a login page presented as a web frontend. Because one of the requirements is to support Single Sign On (SSO), we decided to support Shibboleth[8] as the SSO mechanism. The user can decide whether he logs in using a local account or a Shibboleth account. This is fully conformant with the model described in D4.2.

[6] Simple Object Access Protocol
[7] Representational State Transfer
[8] http://shibboleth.internet2.edu/

One of the core requirements in TAS3 is to apply the principle of Service Oriented Architectures (SOA).

SOA is not simply a technology or a product: it's a style of design, with many aspects (such as architectural, methodological and organizational) unrelated to the actual technology. But each component or part of this Architecture has to be accessible as a web service and needs to provide either a SOAP or a REST interface. According to the requirements, the support of SOAP is mandatory. The REST interface is planned for a later phase of the project and is optional. Since the repository functions as a data provider in all TAS3 use cases, it will be accessed by a web service call. That's why the most important requirement for the reference repository is the requirement that it supports SOAP. All functions provided by the repository need to be provided by a web service interface, which is described using a WSDL[9] document. This criteria was the main criteria in the evaluation of existing Open Source repositories. The result showed that only a few repositories (at least in the Open Source domain) could fulfill this requirement.

Most of the repositories do not have an API (Application Programming Interface) at all and were only accessible through a web frontend. Others were only providing a REST access to parts of its functions. Only two of the evaluated repositories have SOAP and REST support. And one of those two externalizes even all available function as web service (either SOAP or REST). The complete analysis can be found in the next section.

3.2 TAS3 reference repository - FEDORA

3.2.1 Comparison between Fedora and other existing Open Source repositories

We have chosen **Fedora** as our reference repository in TAS3 to store PII. Before we made that choice, we evaluated also other repository systems. In the table below, the reader finds an overview of the evaluated Open Source Repositories with facts about them and the main reasons for choosing Fedora and not DSpace, CDSWare or EPrints. This table links also to the 'Requirements deliverable' D1.2. [3] Under 'Related Requirements' (a row in the table) the reader can find the corresponding requirements number in D1.2.

Name of Solution	**Fedora Repository (and its special Version Muradora)**
Link	http://www.fedora-commons.org/
Access	open source
Functionality	Repository for all kind of data. Accessible through a web service interface. Can be integrated in a SOA.
Limitations with respect to TAS³	Is not aware of **TAS³** secure or trusted communication.
Related Requirements	R.WP8.6
Justification of Selection	- The Fedora repository can be completely integrated in a SOA. - In Fedora all functionalities of the repository are accessible through a SOAP or REST based web service interface. - Moreover, Fedora is Open Source and has a strong community behind it.

[9] Web Service Description Language

	- Muradora is special version of Fedora. It provides a user friendly graphical user interface (implemented as webfrontend). Moreover, it includes a simple 'Policy Editor' to manage policies. Full text search functionality is already included.
Name of Solution	**DSpace**
Link	http://www.dspace.org/
Access	Open source
Functionality	Storage of documents.
Limitations with respect to TAS³	Not all functions available over web service interface. DSpace has well documented scalability issues. The DSpace project team themselves are not addressing the scalability problem, and the code base is not easy to re-architect. DSpace has a complex code base making it difficult to make low level modifications.
Strengths	EPrints has the widest install base, a significant factor in that it goes a long way to ensure its longevity as a fully supported system.
Related Requirements	Partially R.WP8.6
Name of Solution	**CDSware**
Link	http://cdsware.cern.ch/
Access	Open source
Functionality	Storage of documents.
Limitations with respect to TAS³	Not all functions available over web service interface. CDSware does not have a good community around it. The mailing list has had very limited traffic since 2002, which indicates that this project may have sustainability issues going forward. Moreover, it has extremely complex installation steps.
Strengths	CDSware was designed to accommodate the content submission, quality control, and dissemination requirements of multiple research units. Therefore, the system supports multiple workflow processes and multiple collections within a community. The service also includes customization features, including private and public baskets or folders and personalized email alerts. CDSware was built to handle very large repositories holding disparate types of materials, including multimedia content catalogs, museum object descriptions, and confidential and public sets of documents
Related Requirements	Partially R.WP8.6
Name of Solution	**EPrints**
Link	http://www.eprints.org/
Access	Open source
Functionality	Storage of documents.
Limitations with respect to TAS³	Not all functions available over web service interface. The data model causes some scalability issues, although these could be addressed with some development effort. Its method of adding new digital content type can lead to disparate data models and compatibility issues if maintaining multiple systems.
Strengths	EPrints has the most open development community of the four listed candidates.

Related Requirements	Partially R.WP8.6

The main raison to choose Fedora was its feature to provide each function as web service (either SOAP or REST based service). This feature is important in a Service Oriented approach. And this SOA approach is one of the most important requirements in the TAS3 project.

Fedora has been chosen as repository to store PII in TAS3 because of three major arguments:

(1) Fedora is able to be part of a Service Oriented Architecture. Fedora itself is an SOA.
(2) Fedora is based on Open Source software.
(3) Fedora provides most of the functionalities (and in some case even more) that are required by TAS3 for the storage and the delivering of PII.

The major failings of Fedora are as follows:

(1) Does not provide SSO.
(2) Does not provide a user friendly XACML policy editor (only a simple one in the Muradora version).
(3) Does not provide for proof of ownership of linked documents.
(4) Does not provide a **secure** audit trail.
(5) Does not provide a **secure** permanent delete function.

These failings will subsequently be provided by WP4 in an enhanced version of Fedora (or a special version of Fedora called Muradora).

3.2.2 Functionalities in Fedora

Fedora is an Open Source repository founded by Cornell University and the University of Virginia. FEDORA stands for **F**lexible **E**xtensible **D**igital **O**bject and **R**epository **A**rchitecture and is actually available as version 3.1.
Fedora is more than a simple SQL database, because it allows users (e.g. content managers) to store digital objects, which can be complex multi-media content like combinations of images, text, audio and video. Moreover it allows the attachment of metadata to those assets in order to provide improved search functionalities.
But Fedora doesn't only manage the storage or the ingestion of multi-media content; it also delivers contents to users through a variety of technologies. A special feature of Fedora is the possibility to disseminate content dynamically (realized by so called 'Disseminators').
Other features are:
Versioning of data objects, an integrated Policy Decision Point (PDP) to provide policy based access control, XML export and import of data objects, providing a mechanism based on RDF[10] to design interrelations between data objects (semantic web functionality), search functions based on Dublin Core Metadata or full text search, etc.

Fedora provides a lot of functionalities that go beyond any conventional SQL database. At the beginning of the Fedora project, the system was planned as a repository for librarians, who have very special use cases in the librarian sector. Later on other developers extended Fedora in a way that it has become able to fulfill requirements from other domains like Education, eScience or eScholarship, etc.

The next part of this chapter lists some major functions of Fedora. In the annex one can find a technical description of the interfaces and a short documentation on how to use those web methods.

Access over web service interface:
To be a part in a 'Service Oriented Architecture' the used repository must be accessible over a web service interface. Fedora has different web service interfaces for different functionalities. Moreover Fedora provides SOAP as well as REST access. More details on the web service interfaces for 'Management' API**M**, 'Access' API**A** and the 'Resource Index' **RI**Search can be found in the annex.

Flexible data model for data objects:
PII consists of all kind of documents. The repository has to store text, images, audio and video files, documents based on XML etc. Fedora is able to store all those types of documents. Moreover Fedora can integrate external content, which is stored on other web servers.
The stored content items can be annotated with metadata. Per default Fedora supports the Dublin Core as the metadata format. Any other metadata format is possible and can be realized over 'Datastreams' in Fedora.
More details on the data model can be found in the annex.

Versioning of data objects and audit trail:
Each action in Fedora is logged and available in the 'Audit trail service'. A Fedora data object has an attached event history, which shows all modifications ever done to this object. Moreover older versions of a data object can be requested and retrieved by their date/time attribute. Note that this audit trail is not a conventional audit trail that is stored offline in a vault, but rather is data held within the database about the different versions of the data.

Scalability:

[10] Resource Description Framework

Fedora provides support for more than 10 million data objects. Moreover, the Fedora community is working on a federated system of Fedora servers to support a grid of Fedora instances.

3.2.3 Security aspects in Fedora

Fedora provides integrated mechanisms for authentication and authorization purposes.
The authentication mechanism is based on Java Servlet Filters[11], which allow authenticating using different methods like LDAP, JAAS[12] or over a simple XML based user list with user-password pairs. Since Fedora is a web application that is build on top of Tomcat, it also allows establishing a SSL secured connection between the server and the clients. Those mechanisms are simple but still widely used and for instance secure enough for home banking websites and other apps that are relevant to security.

For authorization purposes Fedora provides an integrated PDP (Policy Decision Point) from Sun (http://sunxacml.sourceforge.net/), which handles XACML (**eXtensible Access Control Markup Language) [**4**]** based policies. Those XACML policies can be either repository wide or more fine grained attached to a single Fedora data object. In the second case this is realized by using a special Datastream called XACML. This Datastream is part of every Fedora data object and defines a policy that is only applicable for this data object. When the Datastream is empty, no policy check happens. Otherwise the XACML policy is analyzed by the PDP.

Because of special functions in Fedora, there exists the possibility to define a Fedora-specific vocabulary for referring to Fedora operations and Fedora-specific entities within XACML policies.

3.2.4 Installation Guidelines for Fedora

Please see this website[13] for more details.

[11] http://java.sun.com/products/servlet/Filters.html
[12] Java Authentication and Authorization Service
[13] http://fedora-commons.org/confluence/display/FCR30/Installation+and+Configuration+Guide

4 Mapping the Conceptual Model to Fedora

4.1 The TAS3 ServiceResponder ADPEP web service

The TAS3 ServiceResponder ADPEP is the gateway between TAS3 and the Fedora repository to service incoming requests. It shall allow databases, repositories and other services to provide their services to the TAS3 ecosystem.

In this first year of TAS3 we've implemented adaptors for the Fedora repository and deployed them as web services on top of the web service layer of Fedora. A Fedora server instance is now accessible by TAS3 and provides special functions needed to be usable in TAS3.

In this section the reader gets an insight into all trust and security relevant interfaces provided by the ServiceResponder ADPEP. Moreover, the functionality of the ServiceResponder ADPEP is explained by presenting the two major components of this web service.

4.1.1 Overview

The ServiceResponder ADPEP web service communicates with several other services in the TAS3 infrastructure. This service receives requests from the AIPEP (Application **Independent** Policy Enforcement Point – see Deliverable D7.1. [5]) in Encapsulating Security Layer (ESL) mode or directly from another ServiceRequester ADPEP in Application Protocol Enhancement (APE) mode.

Moreover, it has to negotiate with the 'Trust and Privacy Negotiator' whether a requested service is trusted or not. This 'Trust decision component' is not yet available and will be implemented in WP4.

Other minor components are the so called 'Obligations Watchdog' (implemented in WP4) and a special database to store sticky policies (implemented by WP8).

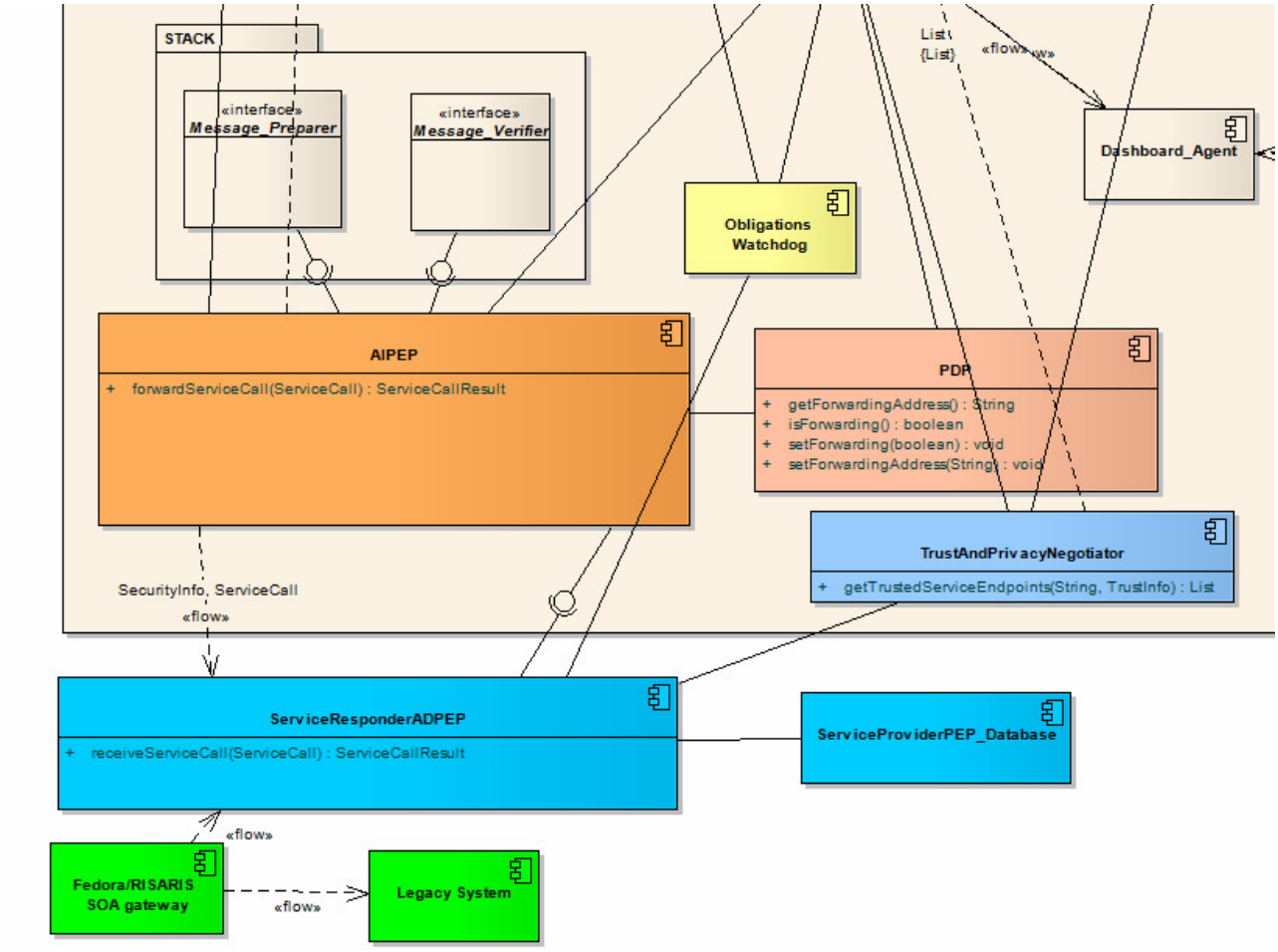

Figure 7: Part of an UML diagram showing the major components on the Service Provider side in ESL mode

Figure 7 only shows a part of all the TAS3 related components, in particular all components that the ServiceResponder ADPEP is talking to. The WP8 components are colored in blue and green. The green colored components are applications, which connect to TAS3 via the ServiceResponder ADPEP.

Beside Fedora other legacy databases will be integrated into the TAS3 world. This task is done by using the Risaris SOA gateway and is described in section 5.

4.1.2 Trust and Security related interfaces of the ServiceResponder ADPEP

The 'Application Dependent Policy Enforcement Point' (**ADPEP**) can be seen as a gateway that provides access to the TAS3 infrastructure for applications like web applications with a web frontend, business process engines, databases or repositories and many other systems, which are either requesting or responding over a TAS3 secured and trusted channel. Since the ADPEP is application dependent, it has to be tailored to each back end application that it is supporting.

The Service Responder ADPEP is the major TAS3 component of the repository on the service or data provider side. It provides special functionalities to transform and store data in TAS3 attached repositories or in legacy databases using the Risaris SOA gateway.

The ServiceResponder ADPEP exchanges information with the

- AIPEP
- Audit Guard
- Trust and Privacy Negotiator
- Obligations Watchdog
- And a special Service Provider Database called ServiceProviderPEP_Database

4.1.3 Functional Components of the ServiceResponder ADPEP web service

One major component in the Service Responder ADPEP is the so called TAS3FedoraHelper library. This library focuses on the integration of Fedora repositories to store PII.

Transformation and (Dis)Aggregation Services:
The library is also strongly adapted to the Generic Data Format (described in deliverable D8.2. [6]), because of transformation services (see section 2.3.), which transform incoming content based on a custom format into the Generic Data Format. This transformation functionality is described in deliverable D8.2.
The other main functionality, which is orchestrated out of the TAS3FedoraHelperLibrary, is the Data and Policy Aggregation and Disaggregation (see section 2.4.). This aggregation and disaggregation service is also described in D8.2.

Connection to Fedora:
The TAS3FedoraHelper library uses Fedora's web service interfaces to communicate with the Fedora server. The mostly used interface is the API-A (Access). But also the API-M for managing purposes and the RI-Interface are used to modify, delete and create new objects and to define relations between them. (RI interface, API-A and API-M are explained in the annex)

Using the TAS3FedoraHelperLibrary:
Currently the TAS3FedoraHelper library provides an interface that accepts a payload in Generic Data Format and a command, which defines the action to be executed in Fedora (currently CRUD: Create, Read, Update and Delete). The next step is to disaggregate the payload into its parts (The Aggregator/Disaggregator service is responsible for this task – see deliverable D8.2. for more details). Those parts define the Fedora data objects, which are ingested into the repository. The interconnections between those Fedora data objects are defined using the RELS[14]-Datastream. This RELS-Datastream is a special datastream that allows the definition of relations between objects using RDF (see annex – detailed description of the Fedora Data Model).

[14] RELATIONS

4.2 The TAS3 ServiceRequester ADPEP web service

As stated in the conceptual model of TAS3 repository (in section 2.1.), the repository shall be able to initiate a request to another service. This service is called ServiceRequester ADPEP.
In deliverable D8.3. we describe a ServiceRequester ADPEP for a client, which initiates a request to a TAS3 connected service.
The ServiceRequester ADPEP in this deliverable is for services or more concrete for repositories, which initiate a request after receiving a first request (either from a client or from another repository).
By this functionality a cascade of requests will be possible and PII from different repositories can be fetched, aggregated, verified and the sent back to the original requester.

In this first phase of the project we focussed on the ServiceResponder functionality. When this task is done, we can provide a solid base for the test pilots to fulfil all major use cases. The ServiceRequester for TAS3 repositories is optional and will be implemented and documented in the next phase of the project.

4.3 Installation Guidelines

Software Prerequisites: The TAS3FedoraHelperLibrary runs on Java (JDK or JRE since version 1.6.X) and is packaged as Fatjar. The web service that encapsulates the library is based on Axis 2[15] and runs on Tomcat 6[16].

Installation Instructions: Deploy the Axis 2 based web service as AAR package in the Axis2 web frontend.

License Information: The software is released under BSD License.

Usage Information: Please take a look at a detailed documentation with screenshots in the annex. The documentation is mostly about the library and its usage. The interface of the web service is described by the WSDL[17] file.

Testing: Each major class (i.e. its functionality) in the TAS3FedoraHelperLibrary has been tested with JUnit. (See section 6.1. for a general overview over our testing approach)
For testing the web service, we've implemented a special test client called TAS3Explorer:

> TAS3 Service Responder ADPEP Test Client – TAS3Explorer
> The idea behind this test client is to get a front-end for testing operations in context with TasObjects and a Fedora Database. For usage with Fedora, one TasObject is encapsulated into one Fedora Object, which gets ingested into a Fedora Database. The Client, called TasFedoraHelper, amongst others, supports following functionality: extracting a package file containing XML files, constructing a tree data structure based on their relationship and ingesting them as Fedora Objects into a database. Moreover, the library supports other useful database operations like removing objects, manipulating certain contents and executing updates on them. It is possible to resolve objects from repository and to export the structure and data back to a local archive.
>
> Find more details on this client in the Annex.

Roadmap for future release: The TAS3FedoraHelperLibrary was necessary to adapt the Fedora repository to TAS3. The plan for the future is to provide a more generic library, which also provides functions for other Service Providers than Fedora.

[15] http://ws.apache.org/axis2/
[16] http://tomcat.apache.org/download-60.cgi
[17] http://www.w3.org/TR/wsdl.html

5 Services provided by the TAS3 Risaris SOA Gateway

Risaris will contribute to the project by making existing data in older or 'legacy' databases and applications available in a TAS3 network. This will include data in databases such as DB2, Oracle, MySQL, Sysbase, ADABAS and so on and applications written in COBOL, PL1 or Natural running in environments like CICS. This will ensure these applications can be accessed with relative ease from the TAS3 environment and can comply with any existing rules or policies about that data while also adding to those policies as part of the TAS3 project in general. To clarify this role in relation to Fedora, any new objects required for TAS3 will be stored in Fedora while Risaris will deal with data that already exists in a system somewhere. It has been agreed that for the first deliverable, Risaris will expose the ePortfolio data available from a MySQL database that Synergetics are creating as part of another project. Ultimately this will prove the concept and can then be applied to the defined use cases for the TAS3 project.

5.1 Requirements for the Project

The TAS3 infrastructure will provide a capability to enable data to be exchanged in a secure fashion using existing standards or standards to be developed by the project. The goal of the project is to provide demonstrators to show how this can be applied in existing environments available today in terms of employability portfolios, health and so on. One of the major issues with this is that a lot of data already exists in IT systems which do an extremely good job stand alone but which are not integrated together.

In order to implement TAS3 for existing systems and by extension in other software systems outside which have not been TAS3 enabled, it is necessary to enable the TAS3 infrastructure to access this existing data in a cost effective manner. It is well know that integration can absorb up to 70% of the cost of any project which would make the cost of implementing TAS3 prohibitive. This is where the Risaris SOA Gateway provides benefit to the project by enabling standards based access to existing data sources in minutes with a product based approach compared to traditional methods which involve many man months of effort to provide proprietary access to those data sources.

Risaris and Koblenz have already proven with some simple scenarios how the SOA Gateway can enable access to a simple form of existing 'legacy' data using the SOA Gateway and Fedora. For the project to be successful, it has been necessary to extend the capability of the SOA Gateway to provide a capability to update these existing databases as well as getting information from these databases.

In addition, the goal is to create an abstraction of the data that is on these databases using the SOA Gateway so that the structure of the existing data does not impact on how the SOA Gateway can show this data which will speed up even further then ability to interface such databases to the TAS3 infrastructure.

5.2 Technical Overview

The key to the speed and flexibility which the SOA Gateway offers to interfacing with these databases is in the standards that are used by the SOA Gateway. Simply put, by adhering to recognized standards, unrelated software systems can be set up to interface with each other via configuration exercise which is extremely quick compared to a develop and build scenario used for Electronic Data Interchange (EDI) previously which takes huge effort and results in a brittle, non standard interface between systems.

The standards used by this technology are as follows:

- XML for data exchange
- TCP/IP for connectivity between systems.
- HTTP as the transport protocol.
- SOAP as the application interfacing protocol.
- WSDL to describe the services offered by the SOA Gateway
- REST as a simpler methodology (compared to SOAP) to invoke the services
- UDDI to keep a registry of available services

It should be noted the TCP/IP and HTTP formed the basis for the Worldwide Web and as such have been around and proven for many years. Some of the other standards are a little 'younger' but have still had extensive deployments and use around the world.

5.3 Technical Details

The SOA Gateway exposes existing data sources using SOAP based Web Services. These services are also available via a REST based interface. This means that this data can be made available to a TAS3 client in a standard way. Using the same interface, it will also be possible for TAS3 clients to update the existing data so it will be a two way process.

In the context of the above, Fedora can also act as a client whereby it needs to create objects that consist of new TAS3 information combined with objects from existing data sources. Fedora can access the SOA Gateway and combine the results with data held within the Fedora TAS3 Repository as required.

Unfortunately the Fedora interfaces currently are not at a point where it can also update data in this way, however, as they are improved, this will also be possible.

In order for the SOA Gateway to become part of the TAS3 network, it will interface using the PERMIS infrastructure created by Kent University. This will provide a standard way to protect any of these existing data sources using the SOA Gateway in a very quick and cost effective manner.

Limitations

When REST is being used, there are a number of limitations to accessing the data via the SOA Gateway as follows:

1. It will not be possible to include updates to the database in a larger transaction as the protocols used do not allow for this.
2. User credentials must be provided using the HTTP Authenticate headers which are limited in their nature.

Note that these limitations do not apply when using SOAP to access the data so this will be the preferred option.

Deliverables

In order to support this effort, Risaris will enhance the SOA Gateway to deliver the following capabilities.

Ability to Create Composite Objects
The SOA Gateway will enable the creation of composite, complex objects based on data in one or more existing legacy tables. This will mean that Abstract objects relevant for the subject matter can be created and the SOA Gateway will enable those Abstract objects to be linked to existing legacy database tables. This will ensure that Fedora and/or the TAS3 infrastructure can use the same object definition to get at different types of data now matter what the format of the data in the legacy systems.

Get Composite Object
The SOA Gateway will enable the return of a composite object using an agreed unique identifier using the REST 'GET' Verb. This will result in the data from one or more legacy data sources being returned in one single, abstract object.

Add Composite Object
The SOA Gateway will enable a composite object to be added using the REST 'PUT' Verb. The SOA Gateway will receive one abstract object and will subsequently add appropriate records to the back end legacy database(s) to represent that object in the existing system.

Delete Composite Object
The SOA Gateway will enable a composite object to be deleted using the REST 'DELETE' Verb. The SOA Gateway will receive a unique identified for one abstract object and will subsequently delete appropriate records from the back end legacy database(s) to remove the representation of that object in the existing system.

Update Composite Object
The SOA Gateway will enable a composite object to be updated using the REST 'POST' Verb. The SOA Gateway will receive one abstract object and will subsequently update appropriate records on the back end legacy database(s) representing that object in the existing system.

PERMIS Interface
The SOA Gateway will interface with the PERMIS system from the University of Kent. This will ensure that existing policies from the legacy system can be complied with as part of the TAS3 infrastructure and extended policies may be created by the TAS3 system and enforced in the SOA Gateway such that more sophisticated policies may be applied to the legacy data as part of the TAS3 infrastructure.

Additional Notes

The work to be carried out in the SOA Gateway will ensure that any abstract objects created may be manipulated using the SOAP interfaces in a similar way to the REST interfaces. This will ensure that as the project moves forward, it will be possible to also use the SOAP interfaces and thus open up the possibilities in terms of including such updates in a distributed transaction, perhaps implementing some of the WS-* security standards and so on.

6 Applied Software Engineering Methods

6.1 Integrating Unit Tests in all WP8 components

6.1.1 Unit Tests Overview

As a software project becomes larger and involves more people, the need for more testing also arises. The complexity will make manual testing time consuming, and the manual tests often test a series of coupled methods (integration testing), so finding the exact location of the error may be hard. Additionally the division of code on certain developers will cause those to only test the code they just implemented or changed, without considering any side-effects. There also may be times when integration testing cannot be done, because a backend, like a database for example, isn't available.

Unit testing strives to solve a lot of the problems mentioned just now.
A unit test tests a singular chunk of code (a unit) by checking if the result of the unit's execution meets the expectations. It also does this automatically, so no manual input is needed, and no one needs to work through logfiles. This reduces the time needed to run a test.

Ideally every unit that can possibly break has its own test, so if code is changed later on, it is possible to find faults in a matter of minutes or even seconds, and the unit producing them will be immediately known. By being able to assert failures quickly, less faulty code is shared with other developers, making everyone's life easier.
Though a unit test can never prove the correctness of a unit, it can show faults. This is especially useful when fixing bugs. By writing a test that will create a case where the original code will fail first, and then fixing the code, that bug will be immediately found should it be reintroduced by some means.

When implementing unit tests it is recommended to abstract from outside resources like databases, since it would either require manual work (e.g. starting the database additionally) or would not be able to be run by all developers. There are several techniques to solve this, the most mentioned would be mock objects, which share an interface with the objects actually coming from the database.

A unit test in itself is a kind of documentation of the program as well. A developer should be able to see how a unit is used, and what it is expected to do by looking at the test. This documentation is also always up to date, since changes in the units behaviour would cause its test to fail.

In conclusion, unit testing is a good way to provide a project with more stability and an overall faster development and testing, however it is not almighty. Unit tests won't catch integration or system level errors; they also can't measure other aspects that a developer wishes for, as would be good performance. Therefore unit testing should always be combined with other forms of testing.

6.1.2 How we test in TAS3

There are several Frameworks for unit testing around, but since we are using Java in TAS3 as a programming language, we decided for the most used system: JUnit[18].
JUnit is a testing framework that belongs to the family of xUnit, which are based on a design by Kent Beck.
All xUnit derivates have a few things in common:

- The framework tests units (methods and classes for java)
- The tests are automated
- The idea consists of fixtures(states to start from), tests, test suites (where tests with the same fixture are assembled) and assertions(functions to verify that the outcome meets the expectation)

Aside from being the most used framework, JUnit has other advantages too, like being supported by most IDEs, especially Eclipse. We are using the most recent version, JUnit 4.

When it comes to actually writing tests there are 2 main approaches one can take:

- **White box** testing tests the course of the code. The paradigm suggests writing tests that execute each line of code at least once. Therefore the source code needs to be known, and the tests make sure that every condition in the code once fails and once passes (e.g. for loops, if, switch-case).
- **Black box** testing tests „classes" of input. The paradigm suggests writing tests that not only input common data, but especially boundaries of the domain, or even data outside of the domain.

Since the code of JFedora and Tas3FedoraHelperLIB is rather complex, we will mostly use the second approach for testing.
Though writing good tests mostly depends on experience, there are some guidelines one can follow:

Each test should follow a general structure:

1. **Set up**
2. **Declare the expected results**
3. **Exercise the unit under test**
4. **Get the action results**
5. **Assert that the actual results match the expected results**

The guidelines are described under http://geosoft.no/development/unittesting.html

[18] http://www.junit.org/

6.2 Security Aware Programming

To make an application secure, there are a lot of areas to be secured. If a software engineer wants to only grant authorized access to an application or services, he will at first want to implement a kind of authorization, may it be a name and password, or a special code, or some other restriction of access. Once he has that he will need to worry that the authorization isn't forged or intercepted, thus he need to look at how secure the transfer of the authorization data is. If you are comparing the data used to access his application to something he stored beforehand, like a fix password, he need to see how secure that data is stored.
Aside from authorization, then comes the security of an application and the code. Especially for web-applications the question „can a user really only access the stuff he is authorized for?" is coming up. And more general, can only that functionality be accessed that is supposed to be accessed. Here come into play programming and architectural guidelines, which may differ according to programming language, or probably the system the program is running on. An example here is being prepared for unusual input, which is a very common security issue.
And last of it, a software engineer might want to secure the code against reverse engineering.

Since we are mainly using Java as a programming language, we will summarize some of the security needs of this language.

Limit Access:
Everything should be declared *private*. The less an attacker can access, the less can go awry.

Limit Extension:
Everything should be declared final. This prevents attackers to just subclass a class and then use its modified version to wreak havoc. An inserted extending class can also implement the cloneable interface. If a software engineer needs the ability for extension, the class should be at least not cloneable.

Initialization:
All Objects and classes have to be initialized before running any method.

Package Scope:
We try to avoid this kind of scope. Because new stuff can be inserted into a package without any restrictions.

Inner classes:
We try to avoid this type of classes. Because they are translated into package scope.

These few points are an initial approach to secure our software components. We are aware, that we have to adapt a more sophisticated guideline, when it is available from the 'Integration Work Package' (WP12 is working on this task).

7 Conclusions and Further Development

Since this is the first iteration in a four year project, the decision we made this year were very important for the further development of the 'Repository Services' in deliverable D8.1.

We made our decision to use Fedora as our standard TAS3 repository to store PII. Based on this reference repository we built a first version of the 'Service Responder ADPEP' web service, which comprehends (amongst other components like the TAS3 Transformation services – see more details in deliverable D8.2.) the so called 'TAS3FedoraHelperLibrary'. This library simplifies access and adapts the Fedora repository to the TAS3 infrastructure. To test this library we've implemented a user friendly and powerful test client. All our libraries, packages and classes are tested via unit testing and wherever we could, we applied secure programming concepts.

The next planned steps are to extend the 'TAS3FedoraHelperLibrary' with more functionality like UNDO-functionality, integration of more transformation services. Moreover we plan to migrate from Fedora version 2.2. to the new version Fedora 3.1. To do this we have to adapt our 'TAS3FedoraHelper' library.
The main task is to implement the ServiceRequester component for a TAS3 repository.

Risaris realized a feasibility study, whether Fedora can be used as mediator layer between the TAS3 infrastructure and their SOA gateway. There are still some questions remaining open, whether this approach is feasible or not. If Fedora is not usable as 'shield' or mediator service between the Service Responder ADPEP and the Risaris' SOA Gateway, we will bind the SOA Gateway directly to the Service Responder ADPEP.

8 References

1. Chadwick, David. Deliverable D4.2. - Repository. 2009.
2. Kellomäki, Sampo. Deliverable D 2.1 - TAS3 Architecture. 2009.
3. Gürses, Seda. D1.2- Requirements Assessment. 2009.
4. Moses, Tim. eXtensible Access Control Markup Language Version 2.0. 2004.
5. Chadwick, David. TAS3 Deliverable D7.1. Design of IdM, Authentication and Authorization Infrastructure. 2009.
6. Dahn, Ingo. Deliverable D8.2. - Software Documentation: System: Back Office Services. 2009.
8. Anderson, Anne. SAML 2.0 Profile of XACML,Version 2, Working Draft 5. s.l. : OASIS, 2007.
9. Sermersheim, J. Lightweight Directory Access Protocol (LDAP): The Protocol. 2006.
10. Kutscher, Michael. Deliverable D8.3. – Client / ServiceRequester ADPEP. 2009

9 Annex

9.1 *FEDORA: Overview*

The Fedora repository is realized as a 'Service Oriented Architecture' (SOA). That means that every major component can be integrated as a web service and has a standardized web service interface (described in the so called 'Web Service Description Language[19]'). This modular approach helps to keep the system more extensible and is more fail-safe than a monolithic application consisting only of one major component.
The next figure shows an overview of all available components in the Fedora Service Oriented Architecture.

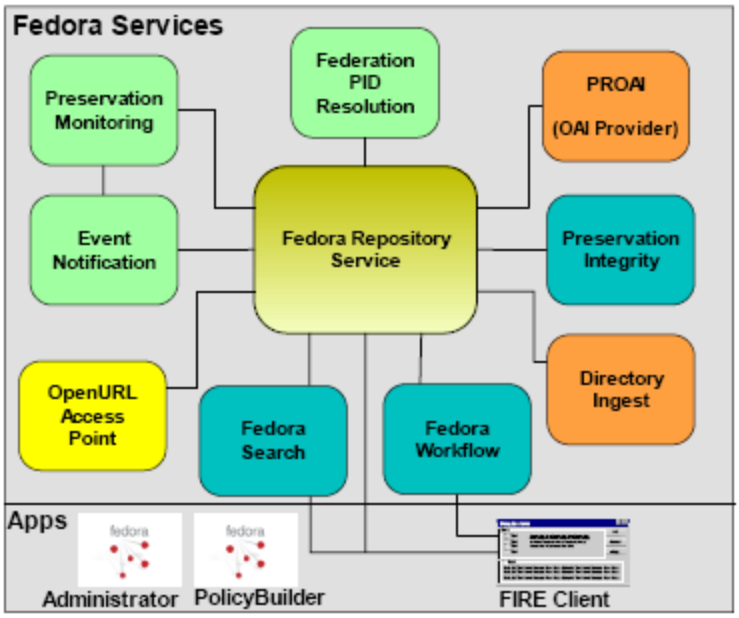

Figure 8: Overview of all available Fedora components [http://www.fedora.info]

In Fedora's 'Service Oriented Architecture' exists a core service called the 'Fedora Repository Service', which is responsible for the main functionality of the system. This service provides two web service interfaces called API-M ('M'anagement of data objects in Fedora) and API-A ('A'ccessing data objects in Fedora). Most of the important functionalities of this core service are exposed via a web service interface. That allows other systems to remotely call Fedora methods and to integrate Fedora in their own 'Service Oriented Architecture'. The simplest way to access Fedora is over a web frontend that makes use of the web service interface (especially API-A). Another possibility is to create a so called proxy client based on the web service description to realize a 'fat client' or desktop client application. 'Administrator' and 'PolicyBuilder' are desktop clients, which use Fedora web services.
Another benefit of using the SOA approach -especially in Open Source development- is the extensibility of the whole system by self-sufficient components (web services), which are implemented by other teams (or single developers) as the core team of Fedora. Every developer in the world can contribute to Fedora and implement his own functionality as a web service. This service can then be integrated (after a revision and tests by the Fedora consortium) into Fedora's 'service world'. If this single service fails, the system as a whole is still intact. The only exception is the Fedora core service, that's why this service is maintained by the main developers of Fedora.
Our approach is to extend this Fedora bundle of services with TAS3 enabled services.

[19] WSDL - http://www.w3.org/TR/wsdl

9.2 ***Fedora web service interfaces***

Fedora's functionalities are exposed via web service interfaces. The main groups of functionality are API-M (M stands for management), API-A (A stands for access) and RI search (Resource Index). Fedora supports both REST and SOAP interfaces.

9.2.1 API-M

The API-M interface exposes management functions in Fedora. After authentication, the client (web service proxy client based on SOAP or a client using REST calls) can execute all important functions to administer data objects in Fedora. This service exposes a set of operations like creating new data objects or modifying them. Deleting and exporting of data objects is also supported.

The next list shows a few major web service operations available in the API-M interface.

ingest: Creates a new data object in the Fedora repository. After ingesting the object a new PID (unique identifier) is created and sent back as return value. The ingest format can be based on the Fedora format called FOXML[20], METS[21] or ATOM[22].

addDatastream: Adds a new Datastream to an existing data object. Each Datastream has attributes, which can be modified by parameters like 'PID' for setting a custom unique identifier, 'MIMEtype' for setting a special MIME[23] type or 'versionable' to enable or disable versioning of the Datastream.

purgeObject: This method allows the deletion of a Fedora data object. Since data objects can have relations to other objects, there may be dependencies to other objects, which would be broken after deletion of an object. To force a deletion, this method provides a flag called 'force'.

modifyDatastreamByValue: Modifies a Datastream in an object. Another method beside the modifyDatastreamByValue is the modifyDatastreamByReference. The difference is, that the former method sends the new content as XML, whereas the ByReference method updates the existing Datastream by providing the location of external or managed content (not only XML based contents).

9.2.2 API-A

The API-A web service interface provides functionalities for accessing Fedora data objects. Fedora allows dynamic dissemination of contents. The API-A interface has functions to discover all available disseminations of an object. Moreover, it provides a simple interface for searching in the repository based on Dublin Core metadata and a method to overview all versions of an object in an object history.

The next list shows a few major web service operations available in the API-A interface.

getDissemination: Executes the service, which is attached to the digital object and delivers the dynamically created content to the requestor. The called service has a signature with a

[20] Fedora Object XML
[21] Metadata Encoding and Transmission Standard
[22] Atom Syndication format: http://www.atompub.org/rfc4287.html
[23] Multipurpose Internet Mail Extensions

web method name and parameters. The parameters are entered as name-value pairs and the disseminator can have several web methods with different signatures. This allows a highly flexible and dynamical dissemination of content.

findObjects: This method allows a simple search based on Dublin Core metadata. The user can refine the search results by entering a simple query with terms or conditions for the search.

getObjectHistory: getObjectHistory lists all available versions of a Fedora data object.

9.2.3 Rest RI Search

Another component in the Fedora system is the so called 'Resource Index'. This index stores all relationships between Fedora objects in a RDF based format. The 'RI Search' service is provided to access the Resource Index and to explore the base relationship ontology. All relationships are represented as a graph that can be queried using an RDF query language.

How are relationships between Fedora objects encoded and stored?

Fedora relationship metadata is encoded in XML (compliant to RDF[24]). The relationship metadata must follow a prescribed RDF/XML authoring style where the subject is encoded using <rdf:Description>, the relationship is a property of the subject, and the target object is bound to the relationship property using the rdf:resource attribute. The subject and target of a relationship assertion must be URIs that identifies Fedora digital objects. These URIs are based on Fedora object PIDs and conform to the syntax described for the Fedora "info" URI scheme[25]. The syntax for asserting relationships in RDF is as follows:

```
<rdf:RDF

  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

  xmlns:fedora="info:fedora/fedora-system:def/relations-external#"

  xmlns:myns="http://www.nsdl.org/ontologies/relationships#">

  <rdf:Description rdf:about="info:fedora/demo:99">

        <fedora:isMemberOfCollection rdf:resource="info:fedora/demo:c1"/>

        <myns:isPartOf rdf:resource="info:fedora/mystuff:100"/>

        <myns:owner>Jane Doe<myns:owner/>

  </rdf:Description>
```

[24] Resource Description Framework [http://www.w3.org/TR/rdf-primer/]
[25] Available under http://info-uri.info/registry/OAIHandler?verb=GetRecord&metadataPrefix=reg&identifier=info:fedora/

```
</rdf:RDF>
```

Figure 9: Example for a RDF definition in Fedora [http://fedora-commons.org]

Such descriptions of relationships are stored in a special datastream in a Fedora object, the so called RELS-EXT Datastream. All RELS-EXT Datastreams are automatically indexed in a RDF-based resource index.

How to search for data objects or relationships in Fedora?

This resource index provides a unified "graph" of all the objects in the repository and their relationships to each other. The Resource Index graph can be queried using RDQL or ITQL which are SQL-like query languages for RDF. The following iTQL query requests all 'FedoraBDefObjects':

```
select $object $modified from <#ri>

where  $object <rdf:type> <fedora-model:FedoraBDefObject>

and    $object <fedora-view:lastModifiedDate> $modified
```

Figure 10: Example for a, iTQL query [http://fedora-commons.org]

The result is a list of comma separated values. Each row represents a single result.

How to access the RI Search API?

The Fedora repository service exposes a web service interface to search the Resource Index. The next paragraph shows and explains the signature of the REST based web service:

http://localhost:8080/fedora/risearch?type=triples

&flush=[*true* (default is false)]

&lang=*SPO|iTQL*

&format=*N-Triples|Notation 3|RDF/XML|Turtle*

&limit=[*1* or more (default is no limit)]

&distinct=[*on* (default is off)]

&stream=[*on* (default is off)]

&query=*QUERY_TEXT_OR_URL*

&template=[*TEMPLATE_TEXT_OR_URL* (if applicable)]

Figure 11: RI Search signature of the REST based web service [http://fedora-commons.org]

The figure above shows the syntax for requesting triples from the 'Resource Index' in Fedora. The user can choose one of two query languages: either SPO (a very simple RDF query language) or iTQL (a full-featured RDF query language supported by Mulgara[26]).

Moreover the delivered result can be provided in 4 different notations: N-Triples[27], Notation 3[28], RDF/XML [29] and Turtle[30].

[26] http://www.mulgara.org/
[27] http://www.w3.org/TR/rdf-testcases/#ntriples
[28] http://www.w3.org/DesignIssues/Notation3.html
[29] http://www.w3.org/TR/rdf-syntax-grammar/
[30] http://www.dajobe.org/2004/01/turtle/

9.3 Fedora Data Model

The following chapter gives an insight into the Fedora data model. This data model was also important for developing the Generic Data Format (see more details in deliverable D8.2.), which is used as an independent format to store person related data. The Generic Data Format is –among others things- adapted to the data model in Fedora. Since the Fedora repository is our reference repository in TAS3 for PII, it is necessary to describe the most important components in the data model.

The major elements in the Fedora data model are so called 'Digital objects' which consist of 'Datastreams' and 'Disseminators'.

9.3.1 Digital Objects

Fedora uses a so called 'compound digital object' design approach which aggregates one or more content data items into the same digital object. Content items can be of any required format (MIME format) and can be stored in the repository or stored externally and just be accessed by reference by the digital object.

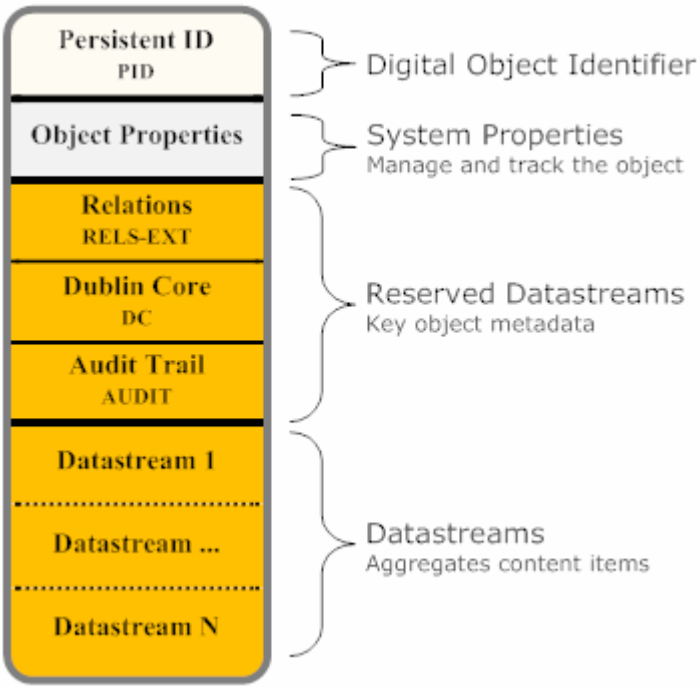

Figure 12: A 'Digital Object' in Fedora [http://fedora-commons.org]

A Digital Object has a unique ID (the so called PID) and some metadata based on Dublin Core standard. Moreover a Digital Object has special attributes to manage and track the object.

Each Fedora Digital Object provides special Datastreams for storing relations between objects (the so called RELS-EXT Datastream) and an 'Audit Trail' Datastream for auditing purposes.

All other Datastreams in a Digital Object are used to aggregate content items.

9.3.2 Datastreams

Datastreams in Fedora store and aggregate all of the content that a Digital Object provides. Each Datastream has an identifier and attributes like:

- 'State': which can be Active, Inactive or Deleted
- 'Versionable': Indicates whether this Datastream is under version control or not.
- 'MIME type': the MIME type of the Datastream
- 'Checksum': used as an integrity stamp for the Datastream content
- Etc.

The Datastream's content can be of the following four types:

- Internal XML Content: The Datastream's content is stored inline with the other data (based on XML) of the Digital Object.
- Managed Content: The content is stored in Fedora's database and the Digital Object has a reference to its storage location.
- Externally Referenced Content: The Datastreams content is stored externally (e.g. on another website or server) and only referenced by an URL inside the Digital Objects XML document. Fedora acts as a mediator and downloads the external content and then streams it to the requestor.
- Redirect Referenced Content: Like in the 'Externally Referenced Content' the Datastreams content is stored externally and referenced by an URL. The difference lies in the access mode. The content is not streamed through Fedora. This is a better mechanism for video streaming, since no mediator is necessary to download the content and then sending it again as a stream to the user.

In TAS3 we will use primarily 'Managed Content' and 'Internal XML Content'. But in the future we will also provide the possibility to integrate external content from other repositories (not only Fedora).

9.3.3 Disseminators

The idea behind the Disseminator mechanism is to add dynamic functionalities to simple, static and 'stupid' data objects.

Disseminators are a special type of Fedora Datastreams to provide dynamical access to Fedora content (meaning Fedora Datastreams). A Disseminator consists of a so called BDEF object that describes the interface to access the disseminator and a BMECH object that attaches the web service functions to the Fedora object.

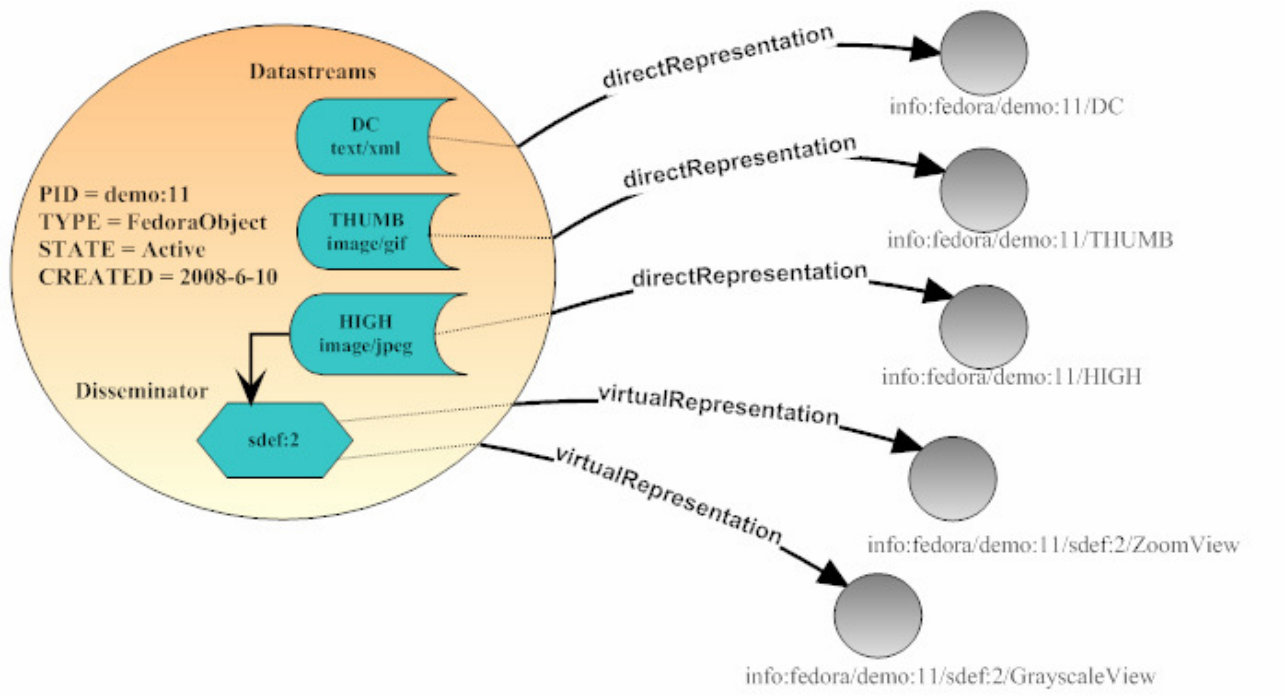

Figure 13: Example of different access methods in Fedora

In the upper figure the reader can see, that Fedora's Datastream can provide different representations of its content. Two of these representations are using a Disseminator. In this case a Disseminator and the underlying imaging web service are used to provide a zooming function and a function to change a picture to a grayscale picture.

In TAS3 we use (among other mechanisms) Disseminators to dynamically provide person related data in different formats. We store all PII in our own Generic Data Format, but are able to transform those Datastreams over the Disseminator mechanism with an attached transformation web service to other formats like IMS-LIP or EuropassCV etc.

9.3.4 Special functionality provided by Muradora (a special Fedora version)

Muradora build a web frontend on top of a Fedora repository with a sophisticated login function and a simple policy editor.

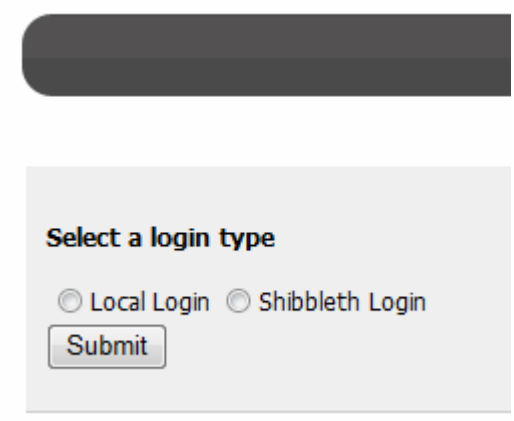

Figure 14: Login selector

After the user selects the Shibboleth Login he gets to the following page, where he can select the right organization and then login.

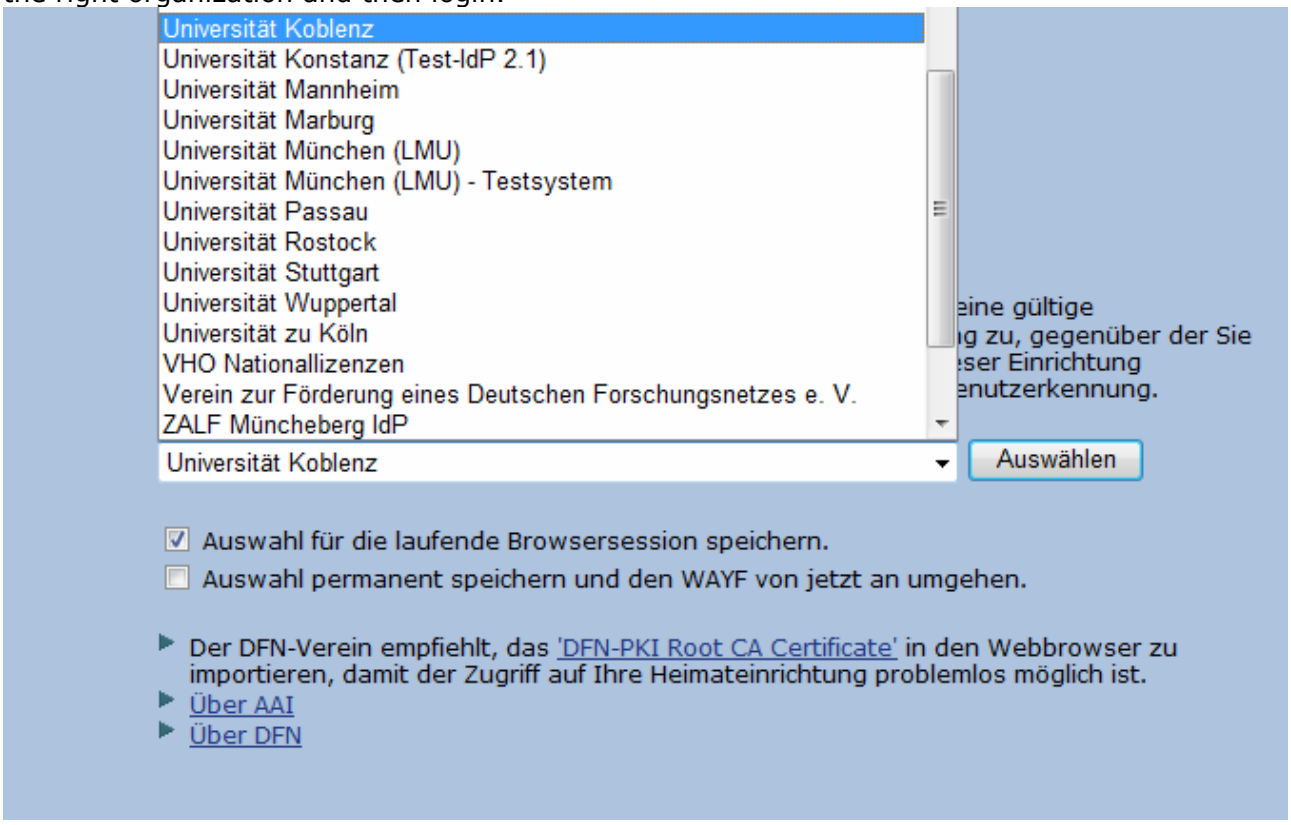

Figure 15: Drop Down list with all available organizations, companies and universities in the TAS3 Trust Network (in this case Shibboleth) domain.

Beside the login there are more contact points where an end user gets into touch directly with the repository over a special web interface. One use case is the definition of fine grained policies, which allow the user to grant or deny access to his digital objects. To define such user centric policies the user needs a policy editor. In the following figure the reader can find a policy editor, which is part of the Muradora[31] project.

[31] https://fedora-commons.org/confluence/display/MURADORA/Muradora

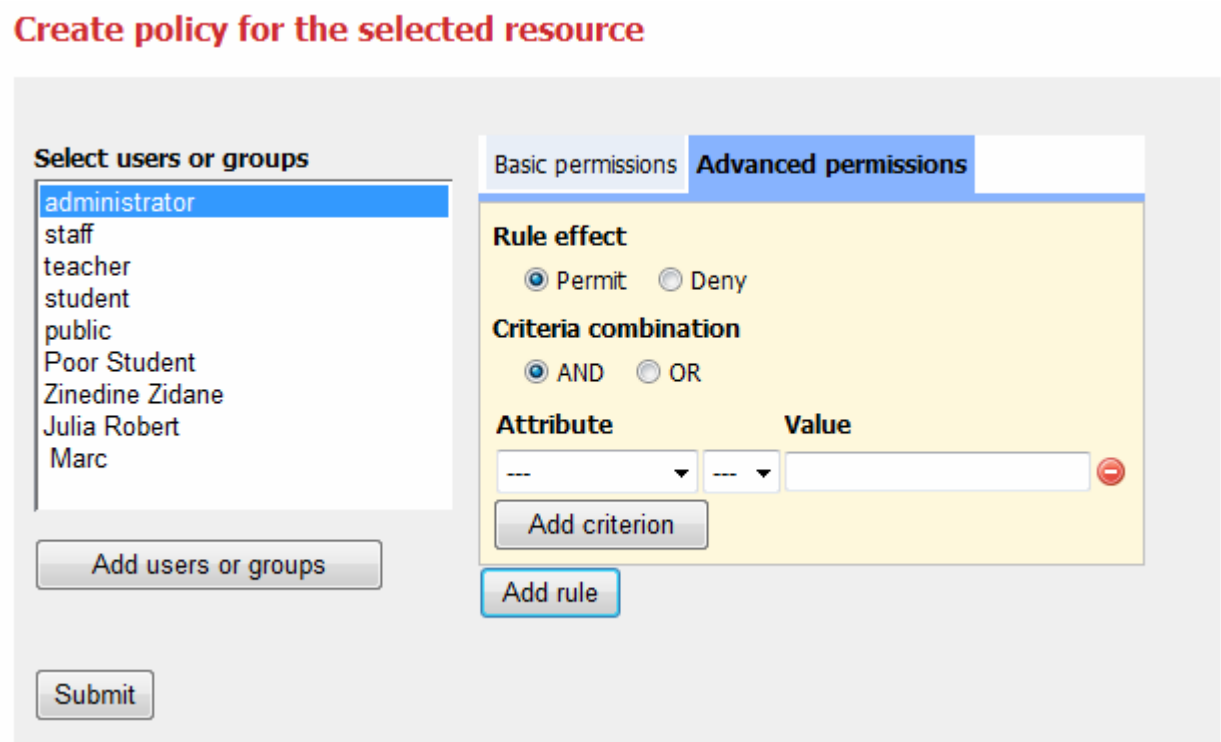

Figure 16: Screenshot of the simple Policy Editor in Muradora

9.4 Design and Implementation of the TAS3FedoraHelper Library

For a better understanding the package structure is reviewed first. Besides the structure itself the responsibilities of the classes inside will be mentioned. Below is a graphical overview of the package structure:

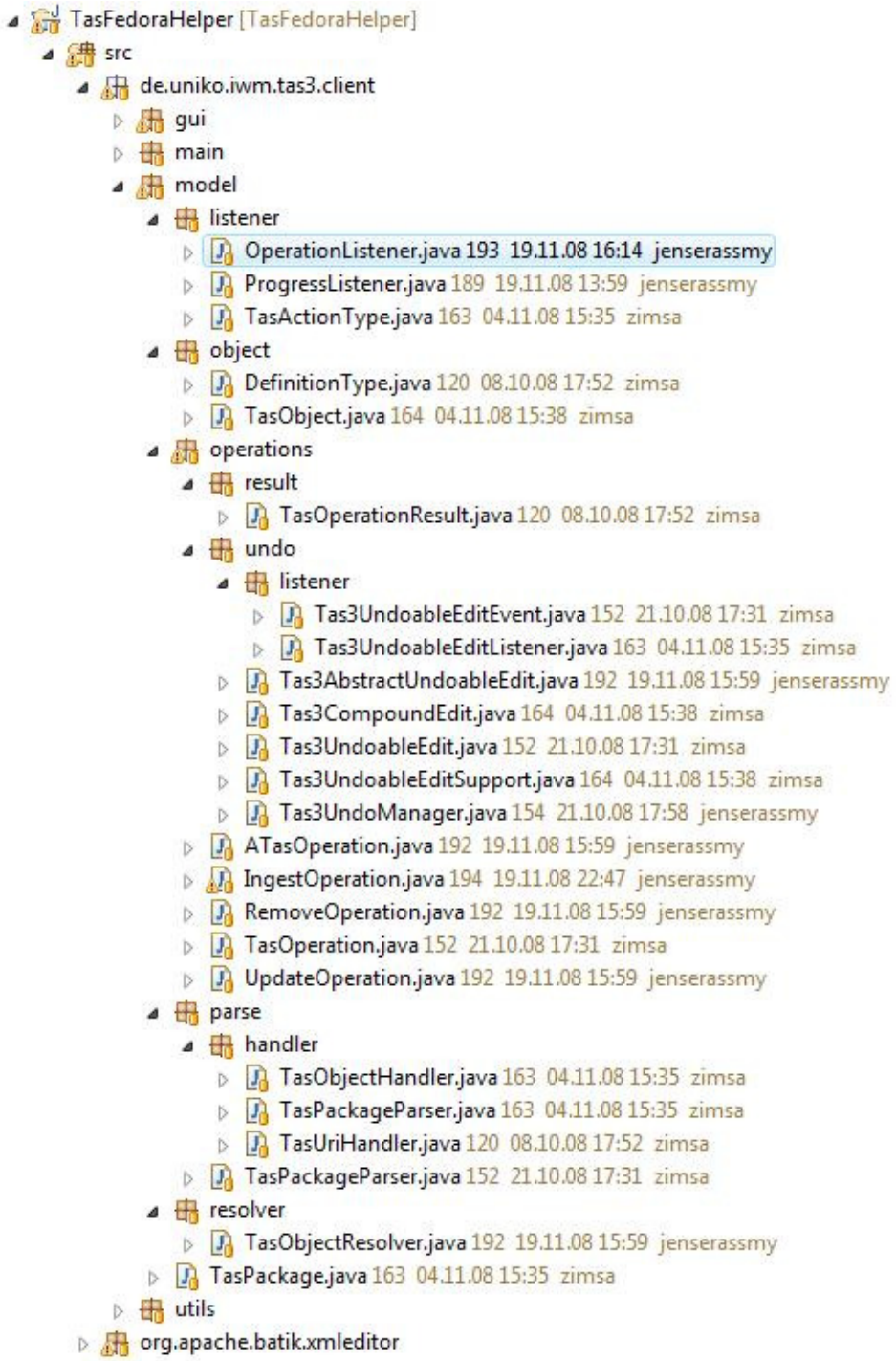

Figure 17: Package structure

GUI connection - de.uniko.iwm.tas3.client.listener

OperationListener:

This class is responsible for running the actions selected by the user. It extends the abstract class ATasMenuListener, which implements TreeModelListener and TreeSelectionListener:

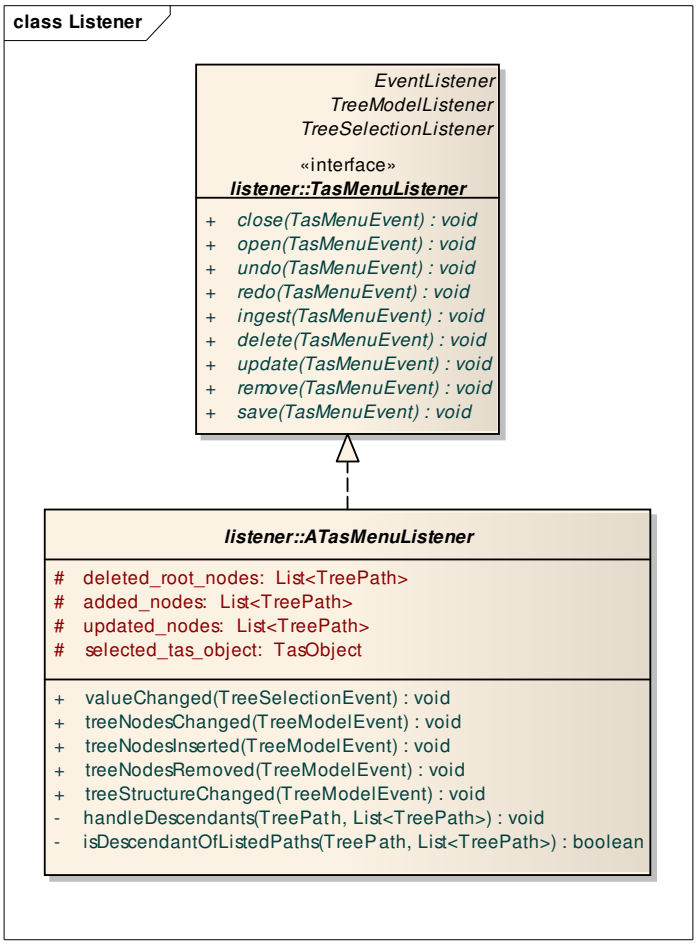

Figure 18: UML diagram of the Listener interface

As shown in the UML diagram, the abstract class keeps track of the selected tree node and updated/selected/added and deleted nodes. For this purpose it makes use of the calls to the listener methods (treeNodesChanged,..). The operations implemented in the extending class OperationListener have access to all needed data.

If anything is missing, information can be held in a singleton object of the type AppProperties which extends Map<String, Object>. The keys are held in the enclosed class Keys, so that they are defined in a central position where a programmer can find them easily.

de.uniko.iwm.tas3.client.gui.**AppProperties**:

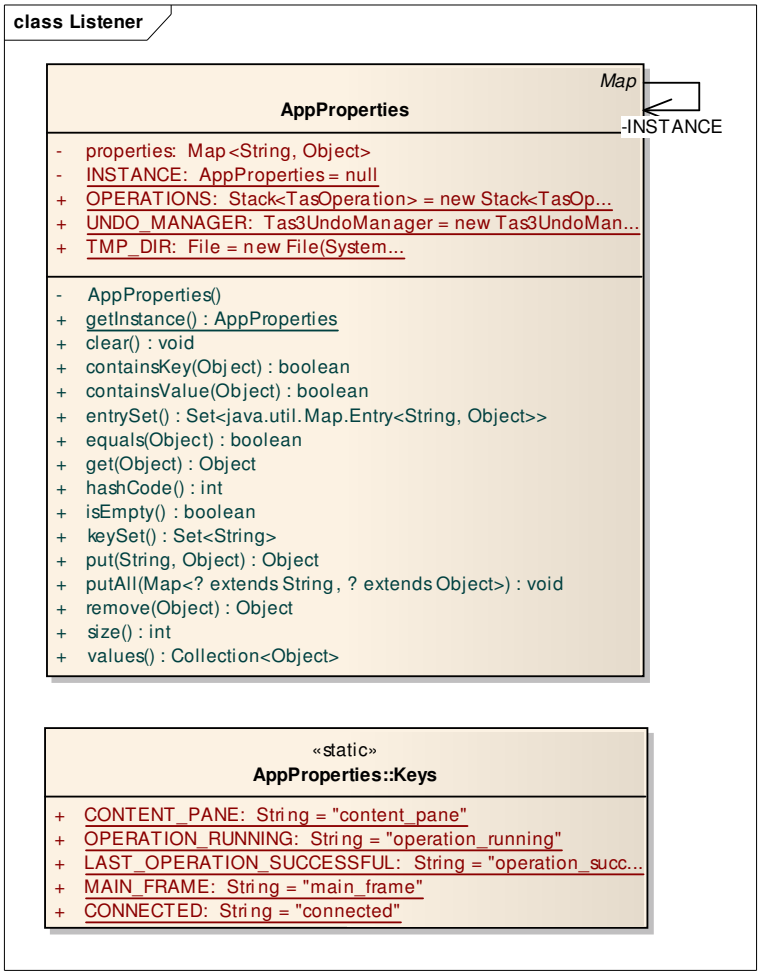

Figure 19: UML diagram of the Listener class

The AppProperties class is a set of a map. The keys used in the map and attributes holding singleton instances of classes are used in different scopes of the code.

The class TasActionType from the first diagram defines actions the user can select together with the appropriate descriptions and text representations to display on a control. (Menuitem, Buttons,...)

The ProgressListener interface is implemented by the dialog showing the progress of user triggered actions. It is the interface between the GUI and the datamodel (MVC). Because of the separation of concerns it is e.g. possible to implement a class for instantiating an object that simply outputs the progress to the console.

This separation was necessary because the code of the operations should be reusable in a web service without greater modification.

9.4.1 Datamodel of the TAS3FedoraHelper Library

Representation of a definitions file

The package **de.uniko.iwm.tas3.client.object** contains 2 classes: TasObject and DefinitionType. A TasObject represents a chunk of xml describing a fact (like the address of the home of a human). This definition makes sense in the context it is defined and the DefinitionType enumeration values should cover the types of facts that can be described.

So every TasObject contains a chunk of xml describing a fact that is given by an enumeration value of DefinitionType.

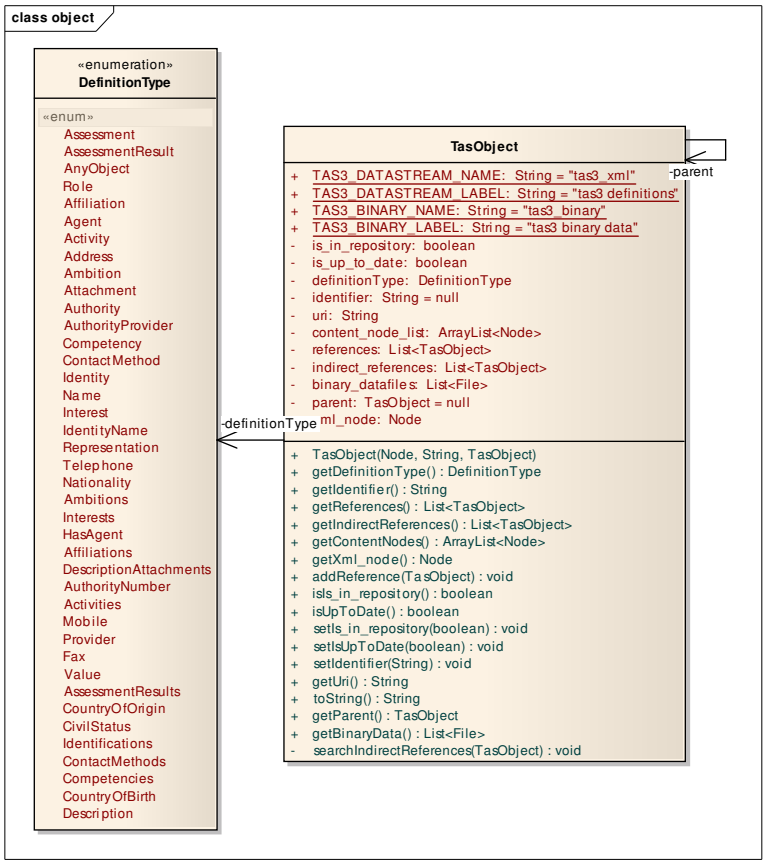

Figure 20: UML diagram of the TASObject class

For the representation of a TAS3 package (in form of an xml document) as a tree, it is necessary to extend the TasObject scheme:

In an xml file containing all definitions (e.g. a transformed portfolio), a definition can have children referencing other definitions. Because a TasObject represents a definition, it is necessary to include references to other definitions when a reference resides in the definition represented by a TasObject. This is implemented straight forward as a list of TasObject instances inside a TasObject.

As a conclusion the root node of a TAS3 objects file is the definition that indirectly references all other defined definitions. After a change in the schema describing a valid TAS3 objects file, the unique identifier of the root definition has to be given in an instance document.

As you can see, there are more aspects as the ones described above:

For managing the definitions in a Fedora repository, you also need the information if a definition that should be processed in a user triggered action is:

- already present in the repository
- up to date

If it is present in the repository, you also have to know the identifier of the Fedora object that represents the definition.

The implementation of the described model would now be able to use TasObject instances as nodes in a tree describing a definitions document. The missing part is a package parser for writing and reading a local format.

9.4.2 TAS3 Package related local I/O operations

At first there was the need for a package format to store the definitions and load them in the Tas3FedoraHelper to develop. The format is defined straight ahead. The package is a zip file containing:

- a file for each definition in the corresponding **tasobjects xml** file
- a file called "**index.xml**" containing the URI of the root element.

The file looks like the following:

```
<index>
    <root>
        <uri>8894CA2F3D689249BBC88B680C153BDA</uri>
    </root>
</index>
```

This format leaves ingest, update and remove operations well-defined and is small-sized. These were the intentions when thinking about defining it.

Since the URI of the root definition has to be given in the TAS3 objects file, a splitter could separate the definitions in files and create the index file. The project doing this is the Tas3XmlSplitter, which will be described later.

For the simplification of the parsing process, an Object of type TasPackage is created first before the parsing is done. The parsing results in the TasObject instances representing the tas package tree.

Let's have a look at the TasPackage class:

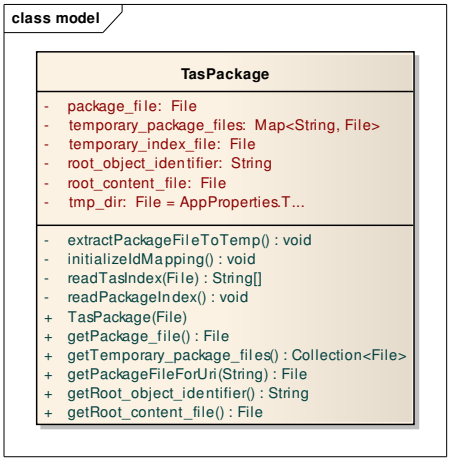

Figure 21: UML diagram of the TasPackage class

An essential simplification is the creation of a map containing the relations of the URIs to the files. So it is easily possible to resolve URIs to definition files while parsing.

Also the files are extracted to the temporary directory when creating the TasPackage and the root file is kept in an attribute after reading the index file.

The class **de.uniko.iwm.tas3.client.parse.handler.TasUriHandler** is used to determine the URI of a definitions file. With the help of a TasPackage object the TasPackageParser can now parse a tas package and create the TasObject instances for representing the TAS package as a tree.

9.4.3 Fedora Repository Operations – Foxml format of a definition

Every definition is stored as an object in the Fedora repository when ingesting. An object representing a definition contains a Dublin Core Datastream, Datastreams for media files included in the package and a Datastream containing the definition as xml (internal xml metadata).

Referenced definitions and parent definitions are stored as "hasPart" and "isPartOf" relations in a Datastream called "RELS-EXT". The "isPartOf" relation is necessary for searching children of a definition with the help of RISearch.

Moreover the Dublin Core Datastream contains the field "type" with the value "tas3RootObject" from the Dublin Core schema whether the definition is a root definition. By that root objects can be searched.

After splitting the files the index file is created with the help of the extracted root URI and the created files and the index file are packaged to a "TAS Package".

Example foxml document of a root object:

```
<?xml version="1.0" encoding="UTF-8"?>

<foxml:digitalObject PID="utils:1476"

        fedoraxsi:schemaLocation="info:fedora/fedora-system:def/foxml#

http://www.fedora.info/definitions/1/0/foxml1-0.xsd"

        xmlns:audit="info:fedora/fedora-system:def/audit#"
xmlns:fedoraxsi="http://www.w3.org/2001/XMLSchema-instance"

        xmlns:foxml="info:fedora/fedora-

system:def/foxml#">

        <foxml:objectProperties>

                <foxml:property NAME="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"

                        VALUE="FedoraObject" />

                <foxml:property NAME="info:fedora/fedora-system:def/model#state"

                        VALUE="Active" />

                <foxml:property NAME="info:fedora/fedora-system:def/model#createdDate"

                        VALUE="2008-09-17T02:44:06.355Z" />

                <foxml:property NAME="info:fedora/fedora-system:def/view#lastModifiedDate"

                        VALUE="2008-09-24T20:14:12.313Z" />

        </foxml:objectProperties>

        <foxml:datastream CONTROL_GROUP="X" ID="AUDIT" STATE="A"

                VERSIONABLE="false">

                <foxml:datastreamVersion CREATED="2008-09-17T02:44:06.355Z"
```

```
                    FORMAT_URI="info:fedora/fedora-system:format/xml.fedora.audit" ID="AUDIT.0"
                    LABEL="Fedora
Object Audit Trail" MIMETYPE="text/xml">
                    <foxml:xmlContent>
                            <audit:auditTrail                    xmlns:audit="info:fedora/fedora-
system:def/audit#">
                                    <audit:record ID="AUDREC1">
                                            <audit:process type="Fedora API-M" />
                                            <audit:action>modifyDatastreamByValue
                                            </audit:action>
                                            <audit:componentID>DC</audit:componentID>
                                            <audit:responsibility>fedoraAdmin
                                            </audit:responsibility>
                                            <audit:date>2008-09-18T14:58:00.093Z
                                            </audit:date>
                                            <audit:justification>modify by jfedora library
                                            </audit:justification>
                                    </audit:record>

                                    <!-- other versions -->

                            </audit:auditTrail>
                    </foxml:xmlContent>
            </foxml:datastreamVersion>
        </foxml:datastream>
        <foxml:datastream CONTROL_GROUP="X" ID="RELS-EXT"
                STATE="A" VERSIONABLE="true">

                <!-- other versions of the RELS-EXT stream -->

                <foxml:datastreamVersion CREATED="2008-09-24T20:14:11.558Z"
                    ID="RELS-EXT.2"    LABEL="Relations    to    other    tas3    definitions"
MIMETYPE="application/rdf+xml"
                    SIZE="2266">
```

```
                    <foxml:contentDigest DIGEST="none" TYPE="DISABLED" />
                    <foxml:xmlContent>
                        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
                            <Description ns0:about="info:fedora/utils:1476"
                                xmlns="http://www.w3.org/1999/02/22-rdf-syntax-
ns#" xmlns:ns0="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
                                <fedora:hasPart rdf:resource="info:fedora/utils:1477"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                <fedora:hasPart rdf:resource="info:fedora/utils:1479"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                <fedora:hasPart rdf:resource="info:fedora/utils:1480"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                <fedora:hasPart rdf:resource="info:fedora/utils:1481"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                <fedora:hasPart rdf:resource="info:fedora/utils:1483"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                <fedora:hasPart rdf:resource="info:fedora/utils:1485"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                <fedora:hasPart rdf:resource="info:fedora/utils:1487"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                <fedora:hasPart rdf:resource="info:fedora/utils:1490"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                <fedora:hasPart rdf:resource="info:fedora/utils:1492"
                                    xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
```

```
                                        <fedora:hasPart rdf:resource="info:fedora/utils:1499"
                                                xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                        <fedora:hasPart rdf:resource="info:fedora/utils:1501"
                                                xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                        <fedora:hasPart rdf:resource="info:fedora/utils:1502"
                                                xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                        <fedora:hasPart rdf:resource="info:fedora/utils:1504"
                                                xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                        <fedora:hasPart rdf:resource="info:fedora/utils:1506"
                                                xmlns:fedora="info:fedora/fedora-
system:def/relations-external#" />
                                    </Description>
                                </rdf:RDF>
                            </foxml:xmlContent>
                        </foxml:datastreamVersion>
                    </foxml:datastream>
                    <foxml:datastream CONTROL_GROUP="X" ID="tas3_xml"
                            STATE="A" VERSIONABLE="true">

                            <!-- other versions of the definitions stream -->

                            <foxml:datastreamVersion CREATED="2008-09-24T20:14:12.313Z"
                                    ID="tas3_xml.2" LABEL="tas3
definitions" MIMETYPE="application/xml"
                                    SIZE="1519">
                                    <foxml:contentDigest DIGEST="none" TYPE="DISABLED" />
                                    <foxml:xmlContent>
                                        <tas3:Identity uri="8894CA2F3D689249BBC88B680C153BDA"
                                            xmlns="http://www.tas3.eu" xmlns:tas3="http://www.tas3.eu">
```

```
                                    <CountryOfBirth>Andorra</CountryOfBirth>
                                    <Nationality>Nederlandse</Nationality>
                                    <CountryOfOrigin>Spanje</CountryOfOrigin>
                                    <CivilStatus>Ongehuwd</CivilStatus>
                                    <Name target="01C3951AEDD0AE4AB8F895F645430F27" />
                                    <Identifications target="BBAA428611DCFC4D92D2B0A1E25E0D63" />
                                    <Representation target="686647F0B9B2B246A51029B0768C92FF" />
                                    <ContactMethods target="7B2E0372805E2E43A0D879F867EF9091" />
                                    <ContactMethods target="91613B09B012F6429D3F285DFC9CA2F1" />
                                    <ContactMethods target="FFF2C953B0D3814198DA08C89B3A080E" />
                                    <ContactMethods target="AB302EC35CBB7044AEEEE561BAF3E7DA" />
                                    <ContactMethods target="0ADC4E6A23B6BC43A7A1ADC882A77981" />
                                    <Competencies target="6273E2FADBDD9842846B5FBB3E36E44D" />
                                    <Competencies target="65169146BE64714198408FA68A9F6E1E" />
                                    <Ambitions  target="goal_01-9BD4555797569641BDD1E1694970BCE4"
/>
                                    <Ambitions target="goal_011-4A277D4C4521114E86DC9B8F4E4CABE0"
/>
                                    <Interests
target="interest_01+869CDB0A25433740A07A34756DFFA5AA" />
                                    <HasAgent target="3273FEAA644C074E920D241FDCF0BC16" />
                                    <Affiliations target="1AD77B70B56F0041A845B197181F817C" />
                                    <Activities                                    target="Activity_01-
4DA880BFEE47534BB71C20418DD1E332" />
                                </tas3:Identity>
                            </foxml:xmlContent>
                        </foxml:datastreamVersion>
                    </foxml:datastream>
                    <foxml:datastream CONTROL_GROUP="X" ID="DC" STATE="A"
                        VERSIONABLE="true">

                        <!-- older versions of the dublin core metadata -->

                        <foxml:datastreamVersion CREATED="2008-09-24T20:14:11.935Z"
                            ID="DC.2" LABEL="Dublin Core
```

```
Metadata" MIMETYPE="application/xml"
                        SIZE="234">
                        <foxml:contentDigest DIGEST="none" TYPE="DISABLED" />
                        <foxml:xmlContent>
                                <oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
                                        xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
                                        <dc:type>tas3RootObject</dc:type>
                                        <dc:identifier>utils:1476</dc:identifier>
                                </oai_dc:dc>
                        </foxml:xmlContent>
                </foxml:datastreamVersion>
        </foxml:datastream>
</foxml:digitalObject>
```

9.4.4 Fedora Repository Operations – The TasOperation structure

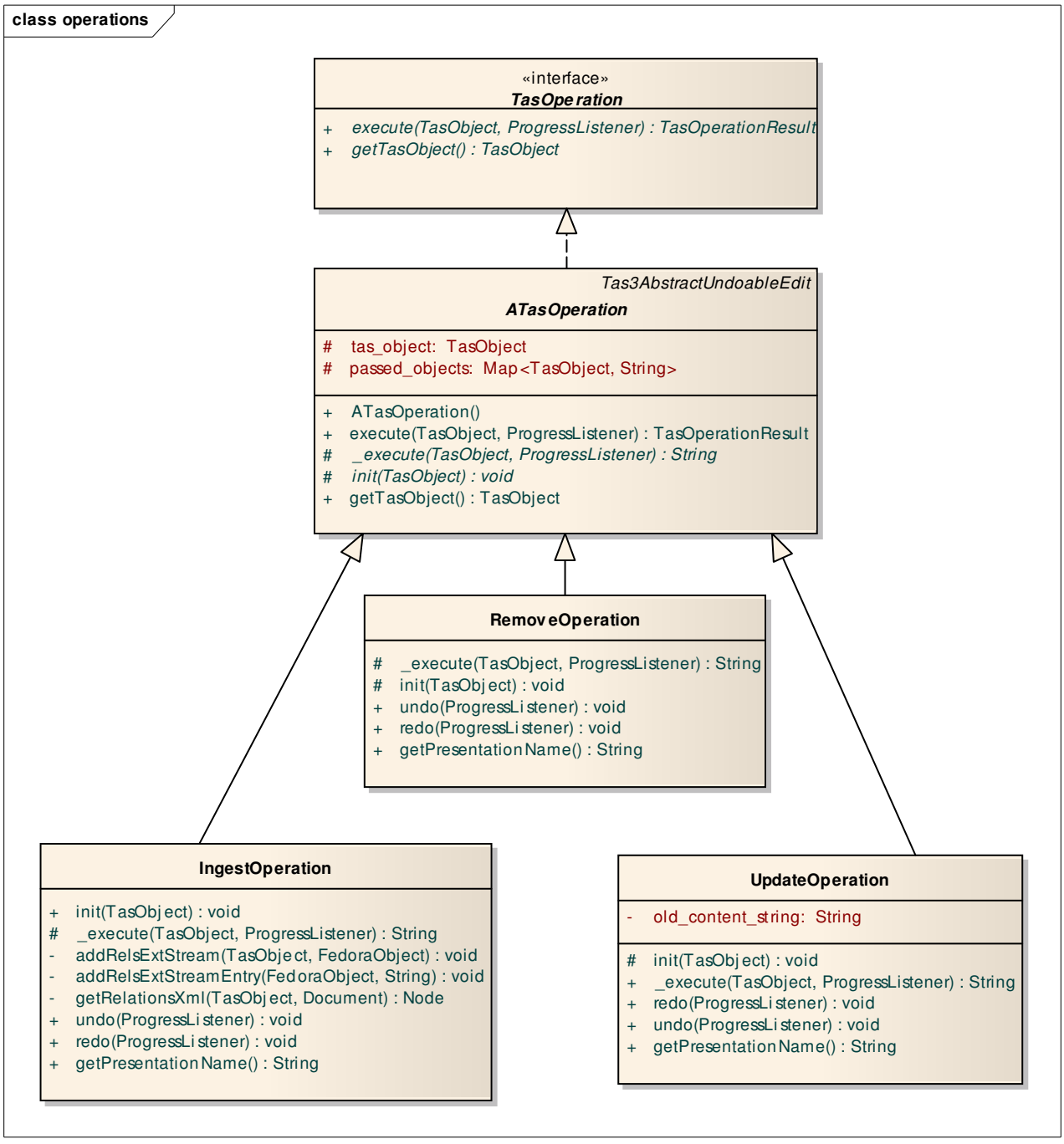

Figure 22: UML diagram of the TasOperation structure

Every TasOperation can be executed via the execute method. The given ProgressListener is informed about processed objects since the progress of every Operation needed can be expressed in that way. (e.g. the ProgressDialog implements this interface)

The class ATasOperation defines the standard methods of a TAS3 operation and the order they are executed. The execute method to be called from outside manages the call of the _execute method to be implemented. Similar procedures to be done for every operation are implemented in the wrapping execute method that every derived class inherits.

Every operation has a redo and undo method also taking a ProgressListener as an argument. And this is why there is a package called **de.uniko.iwm.tas3.client.model.operations.undo:**

The Java API defines a way for managing the undo and redo of operations but does not provide a way to register a progress listener. So there was a need for the customization of the Java API classes. The classes reflecting the necessary changes reside in the named package.

9.4.5 The Tas3XmlSplitter

As described before the Tas3XmlSplitter is responsible for splitting a TAS objects file containing multiple definitions in single definition files and writing the index file.

The current state is that it also packs the content to the package format that was defined first.

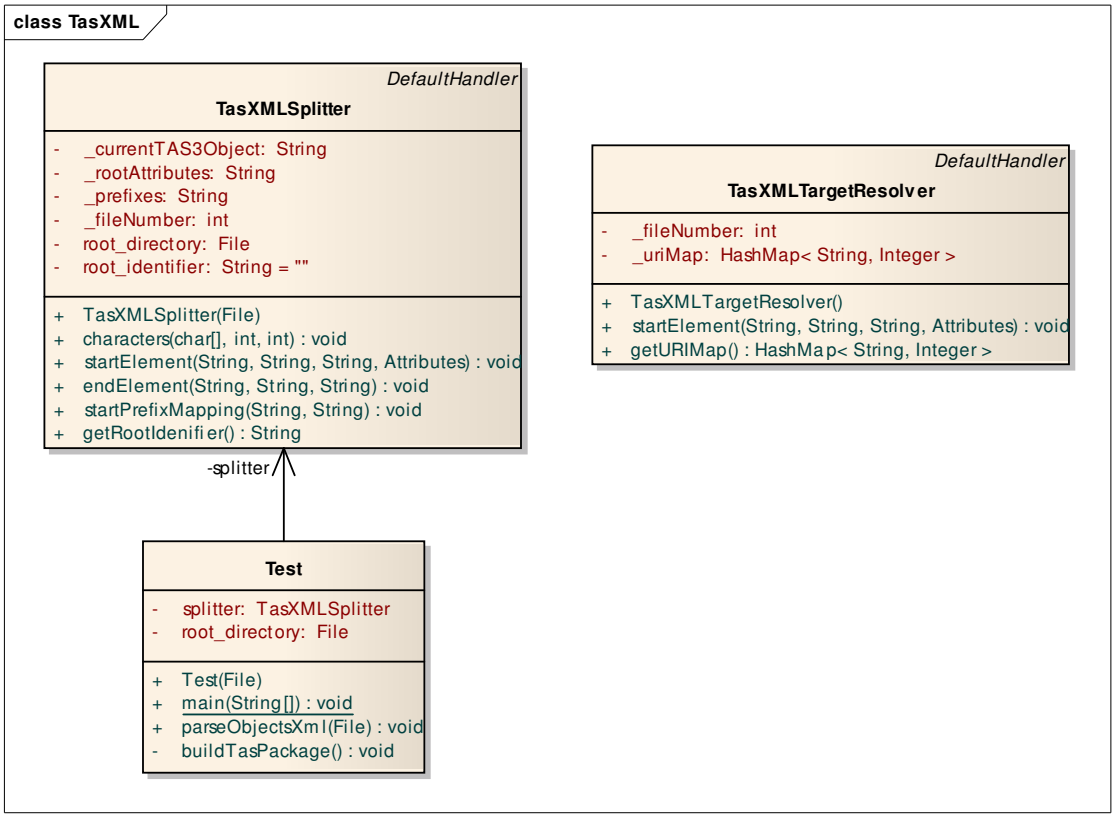

Figure 23: UML diagram of the TASXMLSplitter class

The TasXmlSplitter itself is a DefaultHandler that is registered at a sax parser instance. While parsing, the startElement method of the Handler looks for starting definitions and when appropriate end element are found (call to endElement), the held definitions are stored in their own files.

Also the URI of the root file is extracted (stored in an attribute conforming to the TAS objects schema).

9.5 *More details on the TAS3 ServiceProviderPEP Testclient:*

Services provided by the TestClient-Component

9.5.1 Functions Realized:

Ingest Operation
Ingesting a TasObject into a Fedora Database requires the integration into a Fedora Object, which is uploaded to a Fedora Database. The TasObject content data will be inserted as internal XML Datastream, while the relationship information or optional binary data will be stored as other special Datastreams. After building the Fedora Object, it is checked for references to child objects, which are recursively built. Finally the objects, with a unique ID, are uploaded.

Remove Operation
The Remove Operation supplies the functionality for removing objects, including their children, from a repository. In detail, the relation Datastream of the Fedora object is recursively checked for references. Afterwards reference entries from parents and the objects itself will be purged.

Update Operation
It is possible to modify the content of an object, but it is impossible to add children objects or manipulate the reference structure. To modify the objects content, the corresponding fedora object must be resolved. Afterwards the content can be changed and uploaded back to database.

Search and resolve Objects from repository
Objects can be searched in a repository by their PID[32]. After this, the found object will be recursively resolved and represented in a tree structure.

Undo/Redo for Operations
The integration of an undo operation is not only necessary for UI consistency. In fact, there is a need for reverting failed transactions, foremost by a later implementation as web service.
The operation is realized as UndoManager. Therefore all operations (ingest, remove and update) implement their own undo and redo method.

Load from archive
Local zip files can be read and extracted to a temporary directory. Every zip file must contain an index file, which describes the URI for the root object, several XML files with content information and URIs, as well as directories containing linked binary data like pictures or documents. Based on the extracted XML files the TasObjects and relationships are created.
Export
Changed or resolved objects can be wrote back to zip.

9.5.2 Graphical User Interface

The GUI consists of five components: on the left side is the representation of the TasObject data structure as JTree, the right side shows the content of the selected object: first, a pane for global information, then a table for content and navigation and eventually an XML-Viewer

[32] Persistent Identifier

showing raw content. On the top, there is a menu and a toolbar. If connected, the repository URL is posted in the frame title, otherwise you get a „Not connected"-Information.
All time-consuming operations, like ingesting, removing and resolving, are visualized by a progress bar. After the operation is completed, a dialogue pops up and presents the result (e.g. PID, duration).

Figure 24: Graphical User Interface (GUI) of the text client

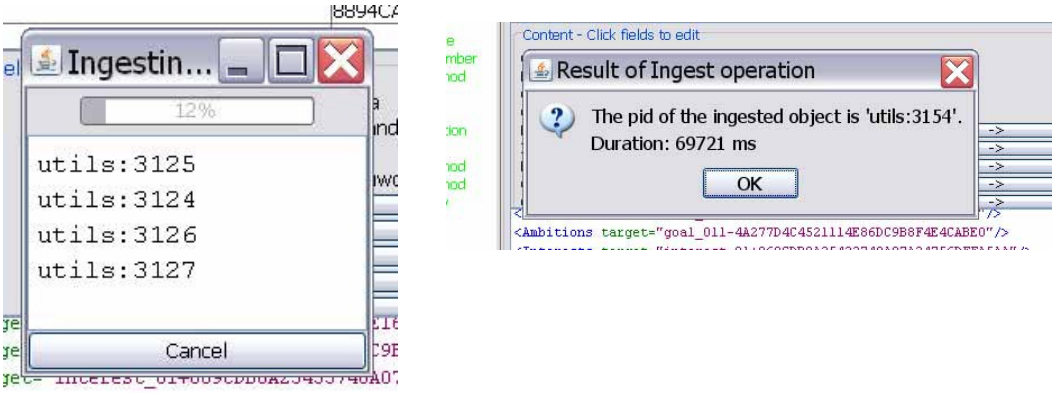

Figure 25: Showing progressbar during time-consuming procedures

Tree View for object structure

The tree view results from the Java JTree component. On the top the root object is represented. Descending the structure, child nodes, standing for the linked objects, are recursively attached. By selecting a node, the view on the right side is updated. Additionally to the „Edit"-entry in the menu bar, one context menu per node is available for executing operations like: ingest, remove or update.

Ingested objects are marked with green colour, others stay black.

Table View for content and additional information

The first table shows information about the selected object. The table represents the definition type, status information, identifier, URI and a link to browse the Fedora object.

The second table contains information about the content. This depends on the XML data. Textual data can be modified and confirmed by pressing enter. Modified objects are marked with a „*". Furthermore, URIs, which lead to the child objects, are visualized as buttons. By clicking, the corresponding node is selected and the view is updated.

XML View for raw XML Content of object

Here the raw XML content is displayed - no editing possible. Changes, made in content table are updated directly.

Menu

The „File"-menu contains actions for opening and saving zip packages.

The „Edit"-menu contains actions for ingesting, removing and updating. As well, an entry for undoing and redoing is shown, if the operation is available. Actions, which cannot be applied, are not disabled.

Toolbar

Figure 26: Test client toolbar

Tree Navigation Buttons

Navigation through the tree: navigating up in structure and back- and forwards in history.

GUI Items for searching and resolving objects

JTextField followed by a Rebuild button: For rebuilding the tree structure of a particular object you have to enter the PID (e.g. „utils:1772") and confirm by pressing Enter or clicking the Rebuild-button.

The Search-button leads to a new dialogue, which presents all ingested root objects. By choosing one, the object will be resolved.

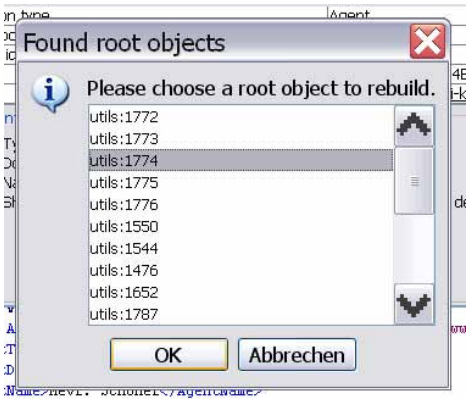

Figure 27: Result list

GUI Items for connecting with repository

Via a JComboBox, you can select a repository or enter a new one to connect with.

9.5.3 Installation Guidelines

Software Prerequisites: The client runs on Java (JDK or JRE since version 1.6.X)

Installation Instructions: Doubleclick the TAS3Explorer.jar

License Information: The software is released under BSD License.

Usage Information: Please take a look at a detailed documentation with screenshots in the annex.

Testing: Each major class (i.e. its functionality) in the TAS3Explorer client has been tested with JUnit. (See chapter 6.1. for a general overview over our testing approach)

Roadmap for future release: The TAS3Explorer depends heavily on the TAS3FedoraHelperLibrary. So each new development iteration of this library must be adapted by the corresponding client.

Amendment History

Rev #	Date	Authors	Changes
1	1 Feb. 2009	Marc Santos, John Power	Initial version by UNIKOLD and RISARIS
2	10 March 2009	Marc Santos	Modified version by UNIKOLD
3	1 April 2009	Marc Santos	Content added, changed layout template.
4	20 April 2009	Marc Santos	Final release for project-internal review.
5	15 May 2009	Marc Santos	Modifications based on comments from internal reviewers.
6	19 May 2009	Marc Santos, David Chadwick, John Power	Modifications and new contents.