europeana
connect

D5.3.1 – Europeana OAI-PMH Infrastructure – Documentation and final prototype

co-funded by the European Union

The project is co-funded by the European Union, through the ***e*Content*plus*** programme

http://ec.europa.eu/econtentplus

Österreichische Nationalbibliothek

EuropeanaConnect is coordinated by the Austrian National Library

ECP-2008-DILI-528001

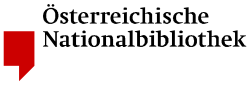
EuropeanaConnect

Europeana OAI-PMH Infrastructure – Documentation and final prototype

Deliverable number/name	*D 5.3.1*
Dissemination level	*PU*
Delivery date	*11/10/2010*
Status	*Final*
Author(s)	*Gilberto Pedrosa (IST); Petz Georg (ONB); Cesare Concordia, Nicola Aloia (ISTI)*

e*Content*plus

This project is funded under the *e*Content*plus* programme,
a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.



EuropeanaConnect is coordinated by the Austrian National Library

Distribution

Version	Date of sending	Name	Role in project
0.1	01.10.2010	Cesare Concordia, Nicola Aloia	
0.2	09.10.2010	Petz Georg	
0.3	10.10.2010	Gilberto Pedrosa	
0.4	15.11.2010	ONB	
0.4	15.11.2010	Jan Molendijk, Theo van Veen	
1.0	07.02.2011	EC, Liferay, public website	

Approval

Version	Date of approval	Name	Role in project
0.4	30.11.2010	Theo van Veen	KB-NL
0.4	17.12.2010	Jan Molendijk	EF Technical Lead

Revisions

Version	Status	Author	Date	Changes
0.1	Draft	Gilberto Pedrosa	01.10.2010	Initial version
0.2	Draft	Cesare Concordia, Nicola Aloia	09.10.2010	
0.3	Draft	Petz Georg	10.10.2010	Some additions
0.4	Draft	Gilberto Pedrosa	12.10.2010	
1.0	Final	VPZ, MK, Georg Petz	07.02.2011	Some formulation changes

Executive Summary

The task 5.3 in EuropeanaConnect aims to provide a solution for Europeana to manage metadata harvesting of thousands of digital heritage content sources across Europe, which results in the REPOX service. IST has led the task of developing the REPOX together with ONB.

The purposes of the REPOX are the management of the aggregator, data providers and their harvested metadata records.

This document describes the specification, design and implementation of the REPOX. The remainder of the document continues with the integration of REPOX in the Europeana ingestion workflow, followed by the refactoring and code review done in REPOX source code.

Table of Contents

1 Introduction

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) http://www.openarchives.org/ has become a cornerstone of the content integration strategy of The European Library (TEL) http://search.theeuropeanlibrary.org/portal/en/index.html. Millions of catalogue records from dozens of providers (primarily national libraries) have been harvested for the TEL portal. Europeana aims to provide access to the content behind the catalogue data – and not only from national libraries, but any provider of digital cultural objects in Europe, from universities to archives to museums. This will lead to a substantial increase in harvesting targets, from dozens to hundreds and eventually thousands. In order to meet the demands of this leap in harvesting scale, this task will provide the implementation of a solution for the administration of available European OAI-PMH data providers and respective data sources. This is an absolutely essential extension of the Europeana portal that will enable large-scale import and thereby rapid attainment of a critical mass of digital content.

The two lines of action that this task will take are: the management of a large number of aggregators and data providers; and the management of the large quantities of metadata records made available by those data providers. As an infrastructure service, this does not benefit end-users directly, but rather indirectly by supporting the integration and efficient management of more content. The direct beneficiaries are the administrators of the Europeana portal, who will be able to manage and integrate content more efficiently (and hence at a lower per-unit cost).

For the management of the OAI-PMH data providers the solution REPOX provides the following functionalities:

- The registration of aggregators, data providers, their collection descriptions (one data provider might make available to Europeana more than one collection), and the configurations for the harvesting of the relating metadata.
- The automatic and manual harvesting of the collections by OAI-PMH (according to configurations and options provided by the data providers and the decisions of the administrators of the central service).
- Monitoring of the quality of service of the OAI-PMH servers, including statistical reporting.

For the management of the harvested metadata records, the following functionalities are provided:

- Support for multiple metadata formats
- A metadata repository service for making the harvested metadata records available to the Europeana
- A scalable solution, able to hold a large number of aggregators, OAI-PMH data providers with a virtually unlimited number of records.

2 Definitions

This document uses the following definitions (some of the definitions are redundant as recognition of other usages that also are common to be found in related documents and bibliography):

- **Data Provider:** an entity that contributes with Descriptive Metadata Sets for Europeana.
- **Data Source:** a service or location, under the responsibility of a Data Provider, from which it is possible to harvest at least one Data Set.
- **Data Set:** a defined group (usually named of “collection”) of one or more Data Records.
- **Data Record:** An instance of a structure of data attributes defined according to a Data Schema
- **Data Model (schema):** The definition of the data elements and the rules governing the use of these data elements to describe a resource.
- **Harvest:** the process to collect Data Sets from a Data Provider.
- **Data Provider:** an entity that has relevant resources on-line and decides to make available their respective descriptive metadata to Europeana, directly through the OAI-PMH protocol, or indirectly by any other mean.
- **Aggregator:** an entity that aggregates Data Sets (Descriptive Metadata) from Data Providers, ideally through the OAI-PMH protocol, with the purpose of making it available to Europeana also through the OAI-PMH protocol.
- **Descriptive Metadata:** data about information objects relevant for Europeana that Data Providers make available for Harvesting.
- **Metadata Set:** by default, the same as a Data Set of Descriptive Metadata.
- **Descriptive Metadata Set:** by default, the same as a Data Set of Descriptive Metadata.
- **Metadata:** by default, the same as Descriptive Metadata.

3 Design of REPOX

This section describes the software architecture and design of REPOX. The design is a consequence of the requirements, both functional and non-functional, which were identified in the documents M5.3.2 “The Europeana OAI-PMH Infrastructure – Updated Specification and Design” and M5.3.3a “The Europeana OAI-PMH Infrastructure – Second Prototype”.

REPOX is an infrastructure to store, preserve and manage metadata sets in XML. It can play the role of a broker or other specific service in a Service Oriented Architecture. It can manage transparently data sets independently of their schemas or formats.

3.1 Service Architecture

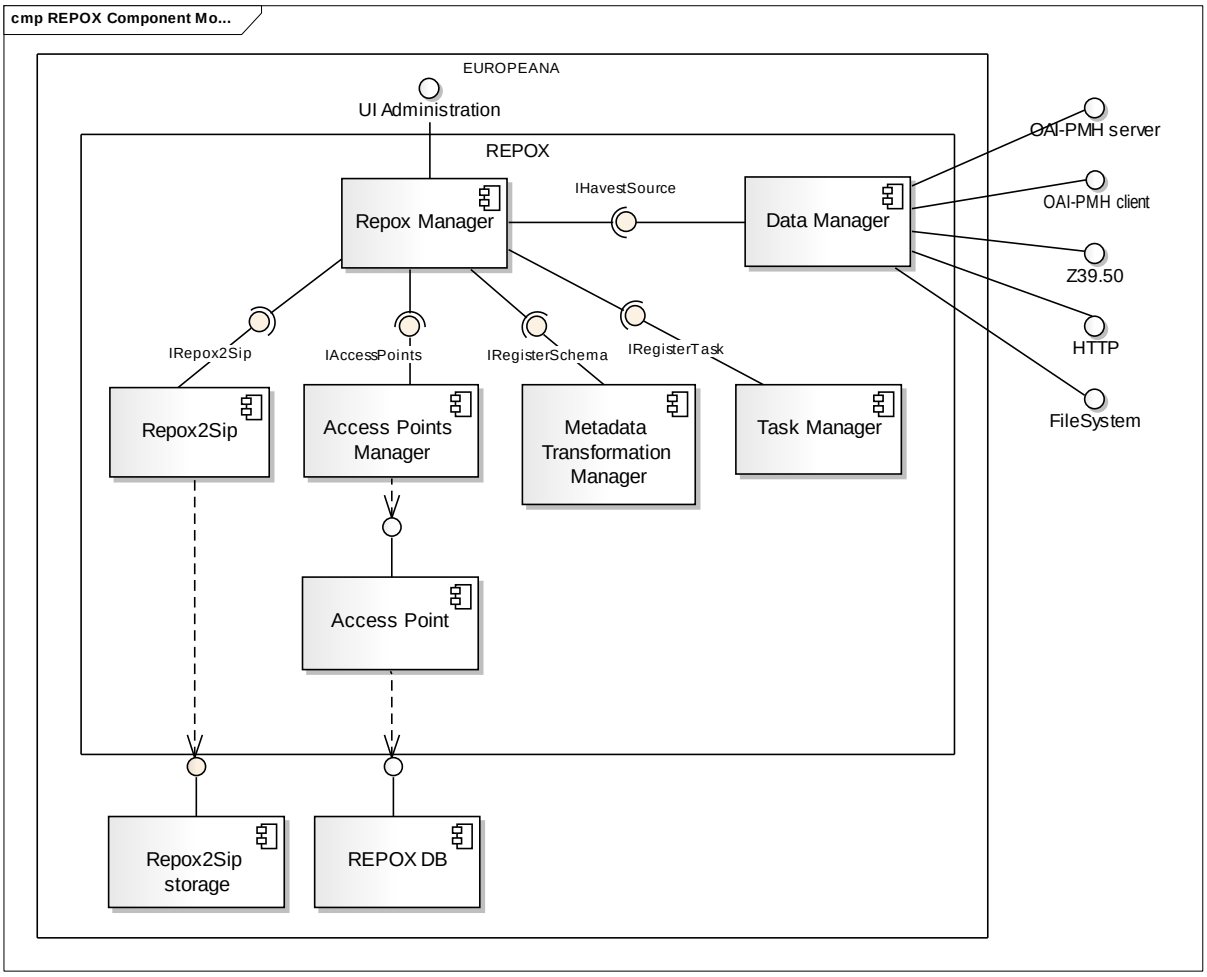


Figure 1 – REPOX context and architecture of components.

The main component of the REPOX infrastructure is the REPOX Manager. The REPOX Manager glues together the other components by managing all the repository processes. It also provides an administration user interface to view the Service status and perform Service operations and an interface to manage Data Providers and Data Sources.

The Data Manager harvests the records from the data sources via the data source interfaces, which may be OAI-PMH, HTTP-get, Z39-50 or a folder in the file Service with files in the format ISO2709, MarcXML, MarcXchange, ESE, or any XML format. The method of choice for harvesting will be an XML Folder or OAI-PMH Client interface. For each Record harvested, the

Access Point Manager creates and stores the indexes of the access points in a database. The Data Manager also provides a set in the REPOX OAI-PMH server for external access to the Records.

The Access Points Manager manages the indexes of the Records. For performance reasons all record content and indexes are stored in the database. An Access Point is an Index to access specific fields of the records in the database. Currently REPOX only indexes the identifiers to access the records content and the timestamp.

The Metadata Transformation Manager is responsible for the registration of transformations between metadata formats (schemas) and for the application of those transformations between specific metadata sets.[1]

The Task Manager is the component that handles tasks with time constraints. There is a background thread that checks for tasks that need to start and launches them when necessary.

The Repox2Sip is a component that is responsible for the integration between REPOX and the Europeana Sip-Manager which is the ingestion tool responsible for creating the Submission Information Packages (SIP). The integration process is described in details in section 3.4.

[1] When available, the REPOX Service will be able to use the XSLT transformations provided by the Europeana Metadata Registry (EuMDR) to perform transformations from each original data format into the ESE profile.

3.2 Data Structures

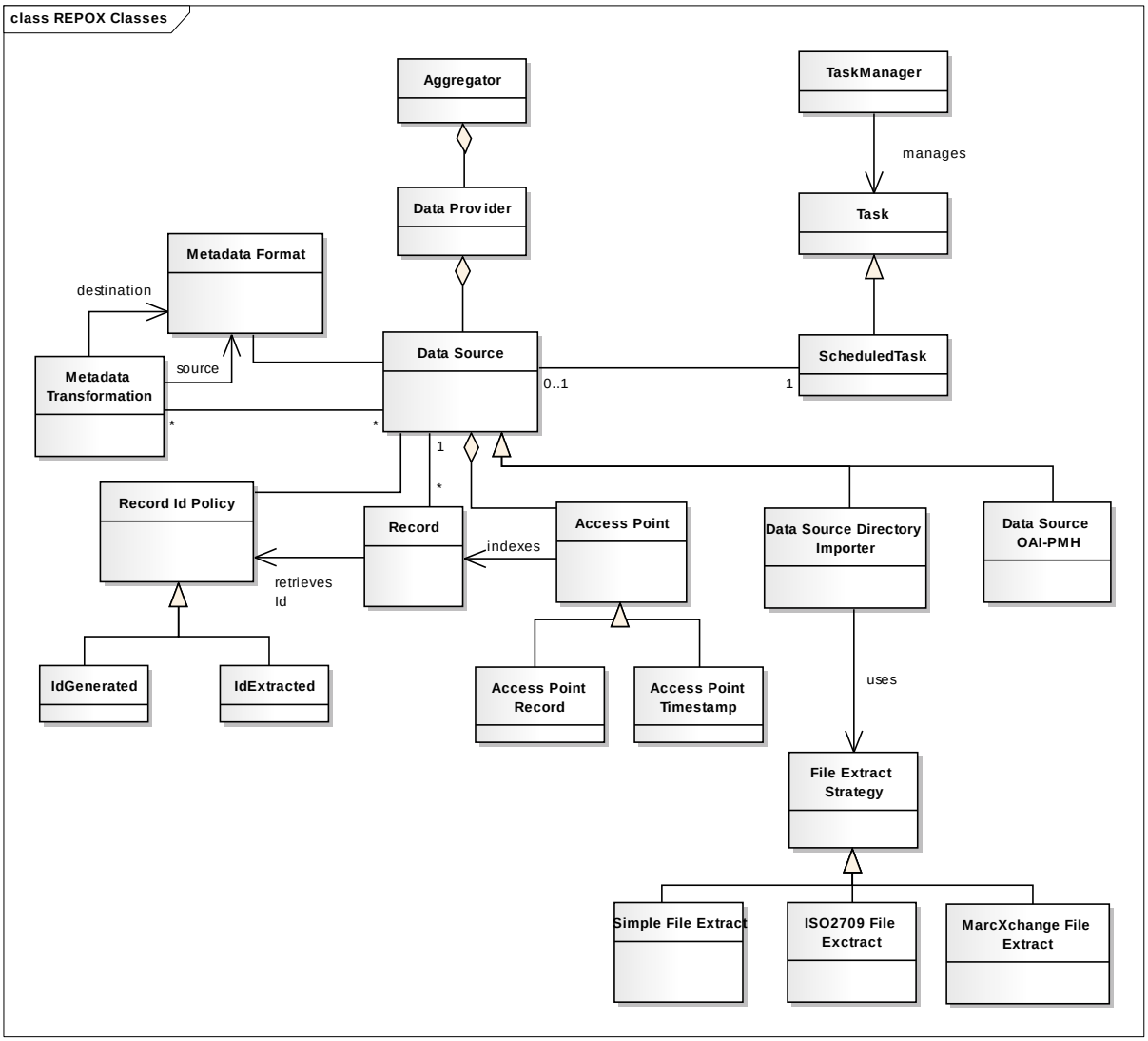


Figure 2 – REPOX domain.

Aggregators are entities that aggregate Data Providers and contain information like name, name code and the homepage. Data Providers are entities with one or more collections of records (record sets) each associated to REPOX by a Data Source. They typically represent an institution (e.g. a Library). Data Sources represent the source of a record set, which is then provided by OAI-PMH.

Data Sources are either OAI-PMH or Directory Importer, the former meaning that the records will be harvested from an OAI-PMH server and the latter meaning a folder in the file Service. To ingest the folders in the file Service, REPOX recognizes three strategies: Simple File Extract, ISO2709 File Extract and MarcXchange File Extract. Simple File Extract is the default method, where there is no processing of the XML records. The only associated logic is validation of the XML. ISO2709 File Extract and MarcXchange specifically target those formats. ISO2709 File Extract requires the file Character Set and the format variant because even though ISO2709 is a standard, some institutions do not follow it exactly. Because ISO2709 is not an XML format and REPOX only handles XML, the format is ingested as MarcXchange because there is no data loss

transforming from one to the other. In the three scenarios the files may be zipped in the file Service and they will be unzipped prior to ingestion.

Data Sources can have associated ScheduledTasks, by scheduling an Ingest of records or an export of the records to the file Service. Those Scheduled tasks are handled by the task Manager. A Task is a managed action in REPOX. Scheduled Tasks are tasks that occur at specific times with a periodicity (unique harvest, daily, weekly and every n months).

Access points (AP) enable the retrieval of the records by more than only their identifiers. For that purpose, access points are associated to Data Sources, to define how to process the pertinent information for indexing. These AP are used by the AccessPointsManager (APM) to extract the relevant data from each record and build the respective indexes. Those indexes are maintained in a relational database for efficiency, as they are not part of the fundamental model.

A Metadata Transformation is a translation between two metadata formats (ex: MarcXchange/Marc21 to ESE). Every Data Source can have any number of transformations. The transformations are stored as XSLT files, even though it is possible to create a visual mapping of them which is stored in an intermediate XML format internally to allow editing. The records can be retrieved by OAI-PMH in their original format (ISO2709 can only be accessed in MarcXchange as specified previously) or any format which has been configured a mapping to. The mapping is performed by request and not stored, because the performance impact is not noticeable.

The record identifiers used in REPOX can be associated in two ways: generated by REPOX or extracted from each record using an XPath expression. The advantage of using extracted identifiers is that it is possible to update just the changes because the records can be recognized by the identifier. In the diagram there is a third option: provided ids in which all records ingested must be sent with their identifier.

3.3 Database Model

In REPOX the records are stored in two databases – internal REPOX database and Repox2Sip database.

3.3.1 Internal REPOX database

There are only two tables in internal REPOX database (Figure 3) for each Data Source: a record table and a timestamp table. The record table stores an internal id, a unique id used for OAI-PMH, a deleted flag and the value in a blob (the value is the zipped XML representation of the record). The timestamp table has the same fields except for the value, which has the date instead of the record XML representation. There are two tables and duplicated fields for performance reasons, when the record is not needed only the timestamp table is used. There is another reason for using two tables: they represent indexes of the record so if it is necessary to have another index for the record in the future, another table would be added with the value for that index.

The tables are dynamically created by prefixing the Data Source identifier. This way every Data Source has its indexes separated, which makes managing (creating, editing and deleting) and accessing the data faster and easier.

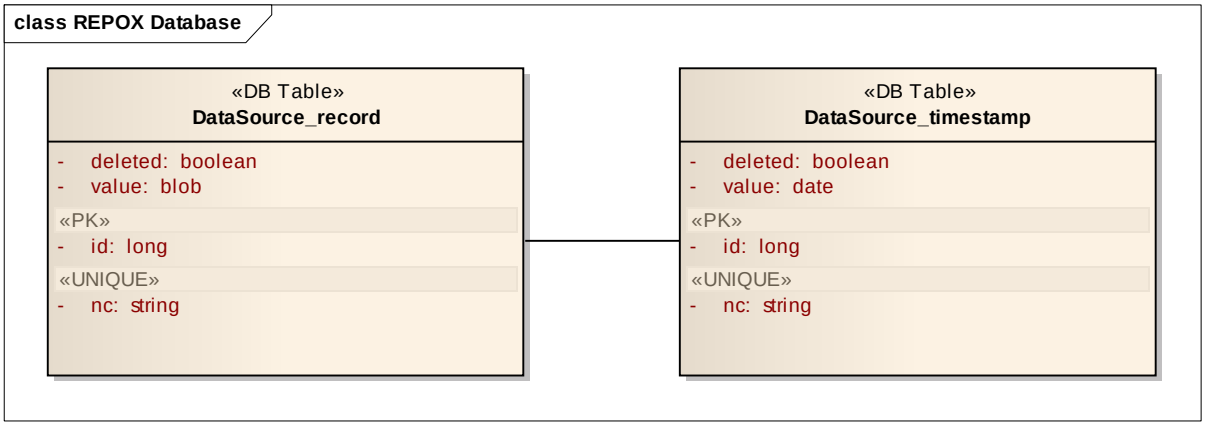


Figure 3 – REPOX database.

3.3.2 Repox2Sip database

Tables in the Repox2Sip database contain information needed to create Submission Information Packages and information needed to synchronize the integration with the Sip Manager.

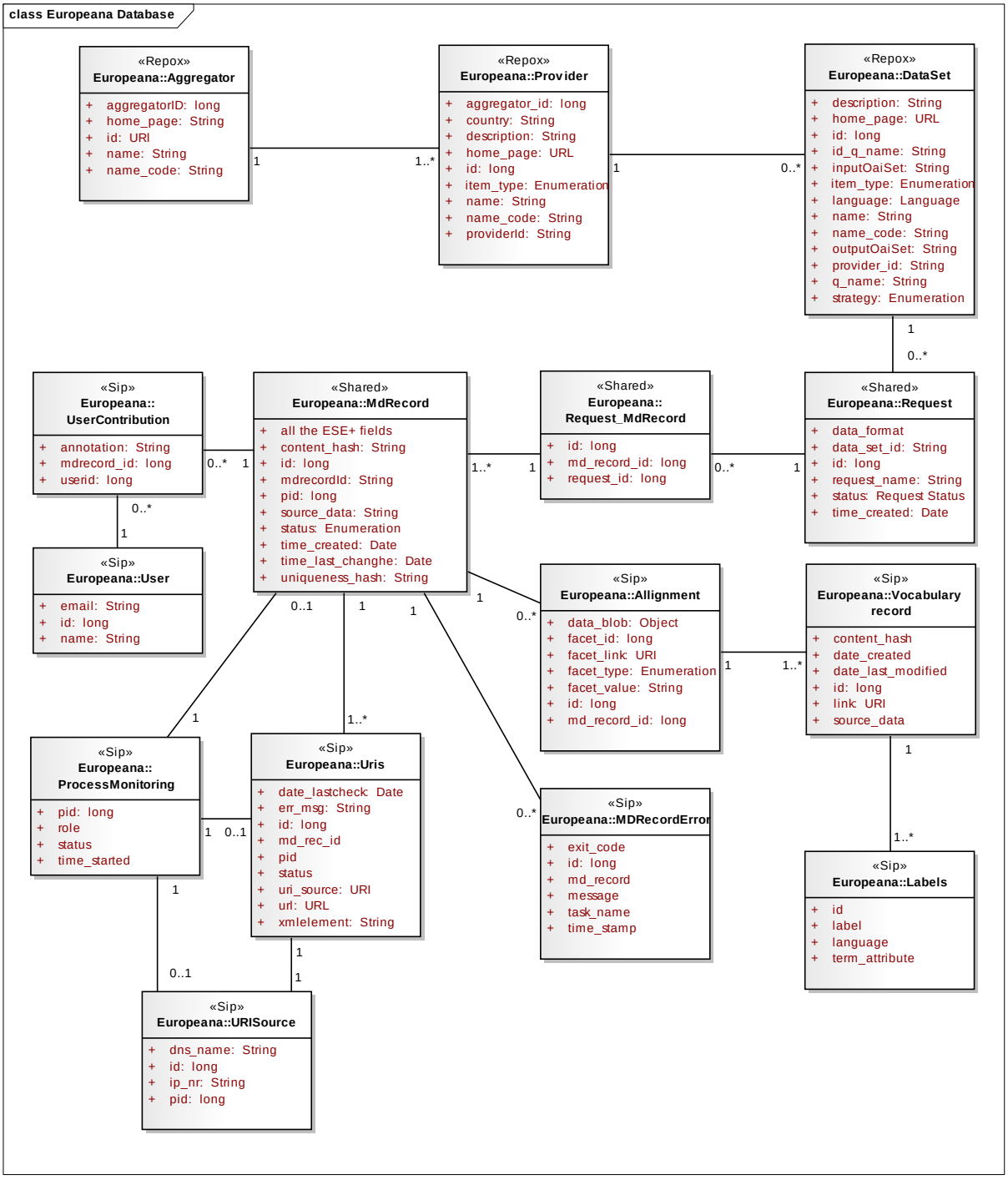


Figure 4 – Repox2Sip data model.

A brief description of tables whose content is managed by REPOX is given below**.**

- **Aggregator table**

The field *name_code* is set by the ingestion operator, currently something like “97”. The field *name* is a human readable name of the aggregator.

- **Provider table**

The field *country* contains the code that identifies the country of the provider, this code can be the two letters ISO code 3166-1-alpha2 (http://www.iso.org/iso/english_country_names_and_code_elements) or the string “eu” if the provider is an EU organization.

The field *name_code* is set by the ingestion operator currently something like “004”.

The *type* values can be:

- Museum
- Archive
- Library
- Audio Visual Archive
- Aggregator
- Research educational
- Cross sector
- Publisher
- Private

- **DataSet table**

The value of the *language* field is a code identifying language of the data set, this can be a ISO 639 two letters code (http://www.loc.gov/standards/iso639-2/) or the string “mul” for multiple languages. Only UTF-8 encoding for harvested files. The value of the field *q_name* is the qualified name used by the provider to identify the root element of the metadata record. The field *id_q_name* is the xpath expression identifying the record id. If the value is empty this means that the REPOX will automatically generate the identifier. The *name* is the name of the data set sent by provider (may be empty), the *name_code* is the name created by Europeana (e.g. 03901_Ag_FR_MCC_joconde). The field *file_name* contains the original request as single file for traceability

The *description* is a REPOX mandatory field and describes the data set; this field is used for OAI server. The strategy field indicates the ingestion strategy adopted by the harvester for a specific data set. Possible values:

- DataSourceDirectoryImporter
- DataSourceOai
- DataSourceZ3950

About *item_type*: currently ESE is the only accepted metadata format but in the future we almost certain extend the harvesting to other formats such as LIDO.

The values of *oai_set* field are defined here:

http://www.openarchives.org/OAI/openarchivesprotocol.html#Set

- **Request table**

This table identifies a specific harvesting for a given data set. It also indicates if this request can be sent to production.

When REPOX is parsing the file the request should be in the state “under construction” and REPOX may abort the request and set the status accordingly, when this process is done the “status” value must be changed in “import completed” and after this point REPOX cannot change the data.

The field *status* can assume the following values:

- under construction – REPOX is creating a new request
- import completed – REPOX ready, sip can take control when ready
- aborted – something went wrong
- sip processing – SIP has found the request REPOX may not any more delete the request
- pending validation sign off – all records for this request completed
- pending AIP sign off
- creating AIP
- AIP completed

- **RequestMDRecord table**

This table links all the metadata records belonging to a given request.

REPOX can only insert records in this table whose request status is “under construction”, if the request is aborted don’t remove links from this table, this task will be done manually.

This table is needed because we want to maintain history of requests for the same data set.

- **MDRecord table**

This table contains the original record and all its refinements. The value of the field *contenthash* will be provided by the Harvester and is used to identify the ESE record.

REPOX may only insert new MDRecord and cannot change or delete existing items.

The two fields that must be actually filled by REPOX are *source_data* with the delivered content dump and *content_hash*, all other fields should have the default values.

The algorithm for generating content_hash is: sha256 hash with all the linefeeds stripped.

The field *status* can assume the following values:

- created – (default) the record is created but not yet processed in any way
- idle – nobody is touching it, waiting for more checks
- processing – a checker is working on this record
- problematic – something went wrong, human intervention might save the record
- broken – record is invalid, some check showed this ESE is not acceptable
- verified – all checks succeeded, could be sent to production

The *mdrecord_id* is an identifier used by REPOX: when no id XPath is provided it is automatically generated by REPOX, otherwise it is extracted using the defined in the *id_q_name* field of the Data_set table.

3.4 Integrating REPOX in the Europeana ingestion workflow

The SIP Manager is the Europeana software component responsible for managing and processing harvested data (http://europeanalabs.eu/wiki/SpecificationsRhineRequirements IngestSipManager).

REPOX and the SIP Manager coordinate their activity by exchanging data via a shared data store. The next paragraph describes the database schema for data and the synchronization protocol for operations.

Figure 4 shows a diagram of the Repox2Sip DB schema using the UML static diagram symbols: class symbols represent tables, class attributes represent fields, and associations represent relationships and their cardinality. Stereotypes above table names indicate the component owning the “write” permissions on the table, for instance the table Aggregator can be modified by the REPOX component while the table URIs can be modified by the SIP Manager. Some tables are shared and can be modified by both.

4 User Interface

During the period of work reported in this Deliverable it was decided to migrate the user interface from the Stripes Framework to Spring MVC.

Both frameworks are based on the Model-View-Controller (MVC[2]) pattern but different techniques regarding its implementation are used. Hence annotations are used instead of interfaces, JSP sites are refactored to Spring Tag Libraries and form validation is applied via JavaBean validation.

In detail this means:

- 17 Stripes Action Beans (Figure 5) have to be replaced by corresponding Spring Controllers. All of the aforementioned 17 classes implement the interface ActionBean[3]. Implementations of this interface respond to user interface events and receive information about the events (usually a form submission). Whereas Spring's web framework since Version 3 is using the help of annotations to handle web requests. The @Controller annotation[4] which indicates that the annotated class is a controller class and can handle web requests is used for this purpose.
- New JavaServer Pages (JSPs) for resolving views have to be created and existing ones have to be adapted.
- Spring form-backing beans have to be applied to represent the forms in the JSPs. For the validation JavaBean validation (@Valid) is used.(Spring 3 includes support for JSR-303[5])

[2] http://de.wikipedia.org/wiki/Model_View_Controller

[3] http://stripes.sourceforge.net/docs/current/javadoc/net/sourceforge/stripes/action/ActionBean.html

[4] http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/stereotype/Controller.html

[5] http://jcp.org/en/jsr/summary?id=303

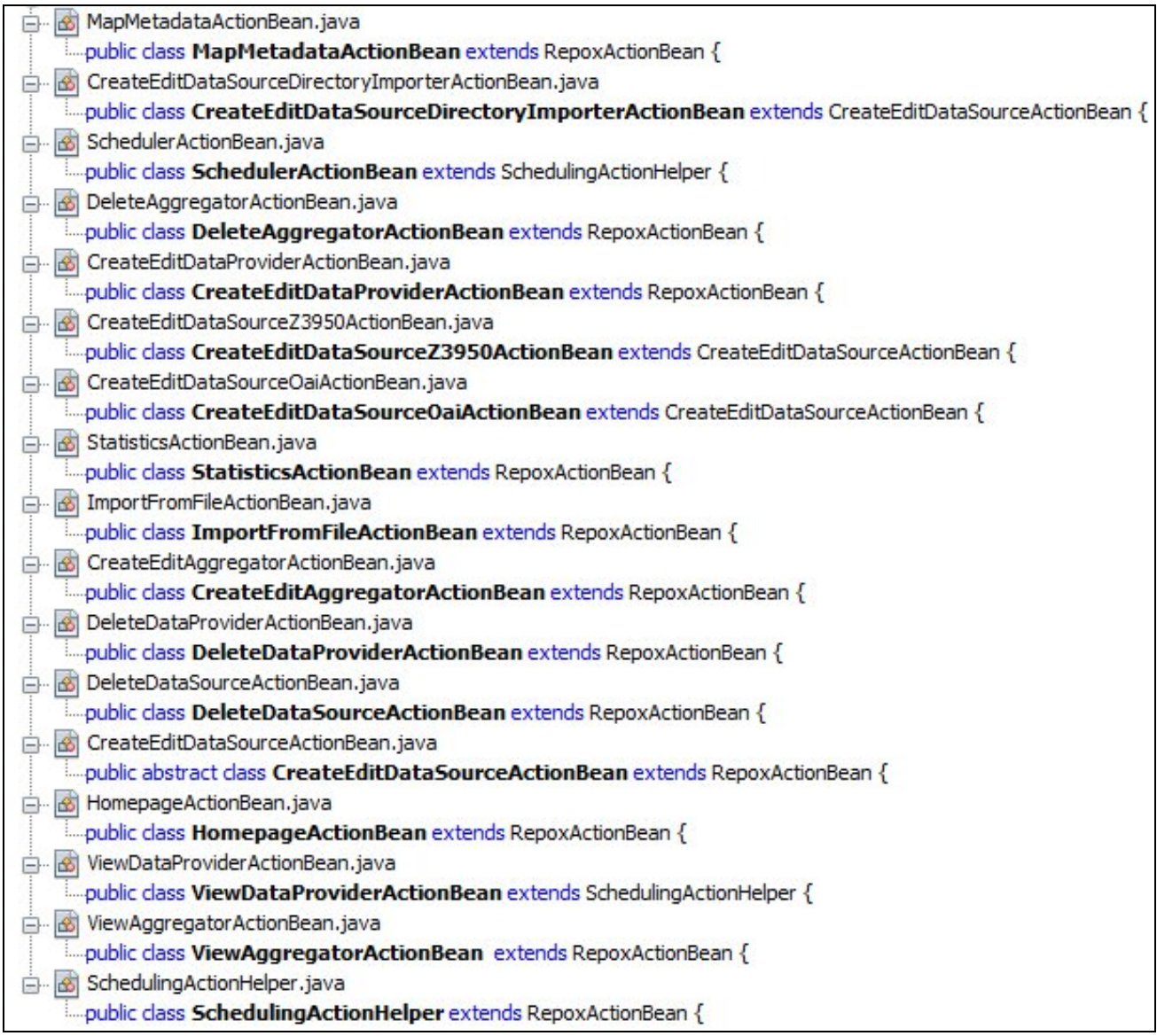
1. 

Figure 5 - REPOX Action Beans.

4.1.1 URLs for CRUD[6] operations

As Stripes will not be used any longer all the URLs will be changed too, e.g. the http request /aggregator/CreateEditAggreator.action?preEdit will be changed to /aggregatorForm.html.

NB: To go from the class name to a URL stripes removes 'ActionBean' (if it is the last part of the class name) and converts it to a path and appends '.action'.

CRUD operations in REPOX have the following syntaxes (taking aggregator as an example):

creating a new aggregator:

http://localhost:8080/repox/createAggregator.html

[6] Create, Read, Update, Delete

reading an aggregator:

http://localhost:8080/repox/viewAggregator.html?aggregatorId=*aggtestr0*

updating an aggregator:

http://localhost:8080/repox/editAggreator.html?aggregatorId=*aggtestr0*

deleting an aggregator:

http://localhost:8080/repox/deleteAggregator.html?aggregatorId=*aggtestr0*

For each CRUD operation a method annotated with @RequestMapping within the Spring Controller and a JSP view has to be created.

4.1.2 Limitations of the intended approach

- No flash scope in Spring MVC: Flash Scope[7] lets you pass a message from one page to the next and only to the next page. Alternatively you could add a message as GET parameter which isn't very clean! Spring offers flash scope only on the Spring Web Flow level (Figure 6).
- Separation of user interface and program logic: There is too much programming logic in the UI-layer. The JSTL tag <c:if> is used too often (Figure 7). This makes the jsp code too complicated.
- JSF vs. JSP: To simplify the development of web based user interfaces JSF should be used rather than JSP. But this means to port REPOX from a Request based framework (Stripes or Spring MVC) to a Component based framework.

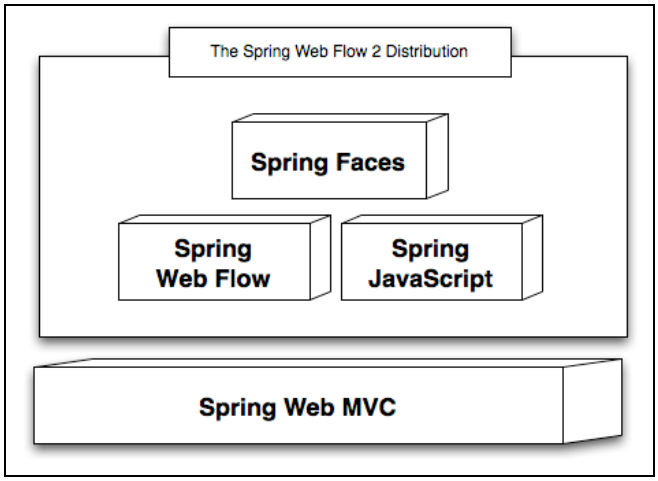


Figure 6 – The Spring Web Flow 2 Distribution.

[7] A FlashScope is an object that can be used to store objects and make them available as request parameters during this request cycle and the next one.

(http://stripes.sourceforge.net/docs/current/javadoc/net/sourceforge/stripes/controller/FlashScope.htm)

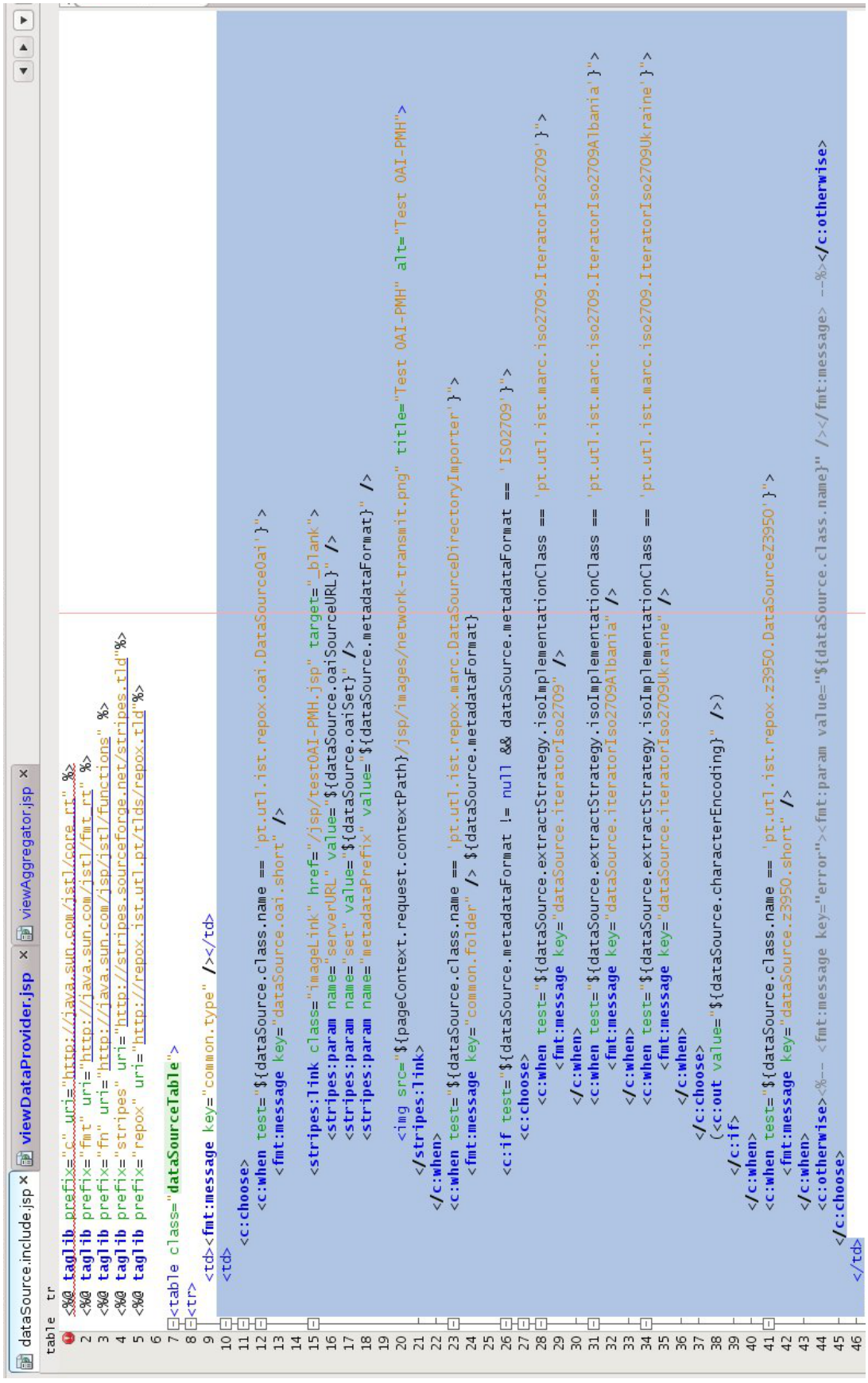


Figure 7 - JSTL tag <c:if>.

4.1.3 Status of the refactoring

The following classes:

CreateEditAggregatorActionBean,

ViewAggregatorActionBean

DeleteAggregatorActionBean,

CreateEditDataProviderActionBean,

ViewDataProviderActionBean

DeleteDataProviderActionBean

(see Figure 8) haven been successfully ported to the *Controller classes in pt.utl.ist.repox.web.spring.

As described in 4.1.2 Spring MVC offers no flash scope.

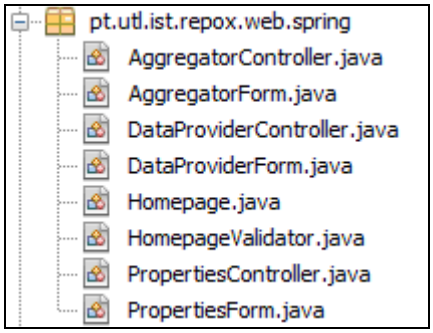


Figure 8 – pt.utl.ist.repox.web.spring.

In package pt.utl.ist.repox.web.spring.session (Figure 9) a Session Bean was created to simulate flash scope. The bean (SessionService.java) is accessed by the controller classes via the Service SessionServiceImpl.java.

Figure 9 - pt.utl.ist.repox.web.spring.session.

5 REPOX source code

The REPOX source code is available at the https://europeanalabs.eu/svn/contrib/repox. This application is using MAVEN[8] according to the Europeana guidelines.

The project is organized in two packages: “main” and “tests”. Inside the main package there are: the java package (with the REPOX, OAI and Europeana classes); the resources (that contains all configuration files and REPOX properties); and the webapps (that includes the documentation and the JSP’s). The main tests of REPOX and Repox2Sip functionalities are in the test package. The tree of REPOX is represented at Figure 10.

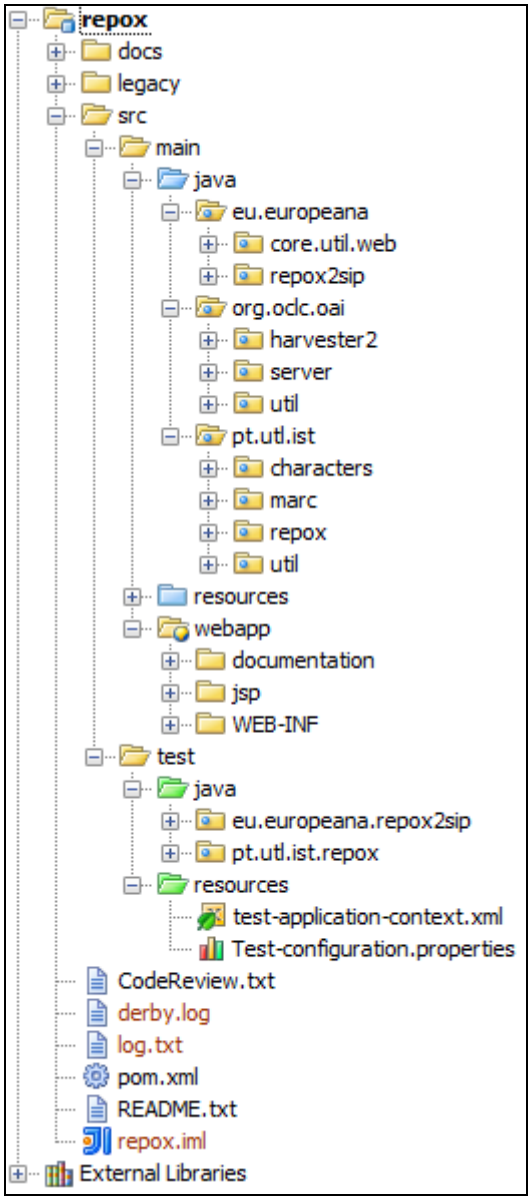


Figure 10 – The REPOX tree.

[8] Apache Maven is a software project management and comprehension tool. http://maven.apache.org/

6 Conclusions and open issues

For the final Europeana Danube release, the main addressed issues were:

- The refactoring of the source code (clean and easy to maintain version)
- Evaluation and improvement of the performance
- The migration from the Stripes interfaces to Spring was started.
- Some others improvements were implemented:
    - enable authorized users to upload to REPOX a file from a local computer and then harvest the file content in REPOX
    - enable authorized users to download harvested files that exist in REPOX
    - improve the "visual feedback" of operations, especially for harvesting (more relevant and dynamic reporting)

Issues for future consideration:

- For pragmatic reasons the current REPOX version uses two databases (Derby and PostgreSQL). In the future would be better use only one database – to avoid data replication and to increase the REPOX performance. In practice, the only implication would be the creation of a new class (specific for Postgres database) that will extend from AccessPointsManager class.
- Comparing the performance between REPOX installation at IST (using only Derby database) and the REPOX installation at Europeana (that uses Repox2Sip and Derby), we realize that Europeana performance is much slower – the performance of Repox2Sip should be improved in the near future.