ECP-2007-DILI-517005

ATHENA

Report on the integration of the plug-in with the Europeana portal

Deliverable number	*D7.4*
Dissemination level	*Public*
Delivery date	*30 April 2011*
Status	*Final*
Authors	*Kostas Pardalis, Nikos Simou, Kostas Raftopoulos, Nikos Skalkotos, Theofilos Mailis, Tasos Venetis, Nasos Drosopoulos*

e*Content*plus

This project is funded under the eContentplus programme[1],
a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

1 OJ L 79, 24.3.2005, p. 1.

Table of Contents

1. Executive summary

Present document constitutes the report on the integration and interoperability of the ATHENA ingestion system with the Europeana Ingestion Office infrastructure and the Europeana portal. Europeana provides integrated access to digital objects from the cultural heritage organisations of all the nations of the European Union. It encompasses material from museums, libraries, archives and audio-visual archives with the aim of making Europe's multicultural and multilingual riches discoverable together in a common on-line environment.

To do this Europeana harvests and indexes the descriptive metadata associated with the digital objects. As there is no one universal metadata standard applied across the participating domains, a set of metadata elements (ESE – Europeana Semantic Elements) has been developed that will allow a common set of information to be supplied to support the functionality desired by the user and needed for the operation of the underlying system.

To provide metadata in the ESE format, it is necessary for contributors to map elements from their own metadata format to ESE. In addition to the mapping it is necessary for a normalisation process to be carried out on some values to enable machine readability. In the initial implementation of the Europeana prototype much of the mapping and normalisation was carried out centrally in the Europeana Office, a work that gradually passed to data providers or aggregators.

The ATHENA project introduced an implementation plan for the ingestion and WP7 provided an appropriate web-based tool that was used for the aggregation and remediation of cultural content metadata belonging to providers. The process guarantees semantic interoperability across numerous data repositories of varied thematic categories, technical features and capabilities, allowing seamless ingestion of diverse content and knowledge and, their subsequent harvesting by the Europeana Office through the OAI-PMH protocol.

2. Introduction

WP7 was responsible for the semantic alignment of metadata schemata that content providers use to annotate their items, to a common, well-defined, machine understandable schema, which in turn will allow for the ingestion of aggregated content in the Europeana portal. Content providers that participated in the ATHENA aggregation constitute mainly of museums and in a lesser extent libraries and archives. As it was illustrated through WP3, and specifically deliverable D3.1 'Report on existing standards applied by European museums', as well as from the various user requirements and analysis studies and meetings that took place within the first year of the project, there is a large variety of metadata standards used throughout the providers. There are a limited number of key standards, and they are used extensively throughout Europe and indeed the world. These are often suggested as best practice but, from the evidence of the WP3 survey, there is still a long way to go to achieve interoperability. National standards in some countries are also a factor that needs to be taken into consideration. There is also often the case where providers bypass the semantics of a standard through proprietary rules and conventions that are not sufficiently documented or semantically defined, resulting in misinterpretations that can not be straightforwardly detected. Finally, there is a vast variety, with respect to the level of detail, in annotation, ranging from the use of a small flat-structured set of elements to defining and using complex schemata that support various levels of annotation, item and concept relations and connections with controlled vocabularies and thesauri.

In this context, joint efforts from WP3 and WP7 have identified the need for an intermediate standard in ATHENA that would serve as the point of interoperability between the providers, as well as between the project's repository and outside sources. These efforts led in the adoption of LIDO (Lightweight Information Describing Objects), an XML Schema for contributing content to cultural heritage repositories. LIDO is the result of a joint effort of the CDWA-Lite, museumdat and Spectrum communities and meets requirements for publishing metadata from all kinds of cultural object classes. It integrates the relevant concepts of the aforementioned schemata into one single schema addressing several important issues that include an event-oriented approach, refined subject element, full support of multilingualism, revised handling of display and index elements and, revised identifier handling (for semantic web needs). In the light of recent and future Europeana developments, LIDO also allows for the contribution of metadata along with digital representations of items, delivers information in a 'self-contained' way, includes links to original repository, distinguishes between display and indexing elements and provides references to controlled vocabularies and authorities.

The adoption of LIDO, coupled with the loosely defined providers' input schemata, led WP7 to design an aggregation and mapping workflow (see D7.1 and Section 4) that allows for an elaborate, visually guided ingestion of metadata in the repository. The fundamental principles include the disassociation of input metadata from existing metadata standards in order to avoid ambiguity over interpretation and, the ability to create and manage transformations that will apply to the actual metadata records, which subsequently (re-)define the input schema in a semantic, machine understandable way, based on its mapping to LIDO.

The web service aims to provide a user friendly ingestion environment that allows for the extraction and presentation of all relevant and statistical information concerning input metadata together with an intuitive mapping service that illustrates LIDO and provides all the functionality and documentation required for the providers to define their crosswalks. Transformations are editable and reusable and can be applied incrementally to user input while providing, throughout all steps,

best practice examples, previews and visual indications to illustrate and guide user actions. One of the key capabilities lies in the ability to semantically enhance user metadata through conditional mapping of input elements using respective transformation functions (e.g. concatenate) that will allow for the addition and enrichment of semantics even when those are not specifically stated in the input.

In this context, the tool also determines the operational work flow processes to bring the amalgamated content of the partner museums into Europeana and to create, manage and execute, with the European Digital Library Office, the implementation plan to ensure that the content is visible in Europeana. The platform supports exporting of the aggregated metadata to several established standards concerning presentation and archive management. Primary effort is directed to the transformation of the aggregated content to the Europeana Semantic Elements schema and the deployment of an OAI-PMH repository to facilitate harvesting by Europeana.
The ingestion tool is based on a metadata aggregation platform that is developed and maintained by the leader of the WP, IVM Laboratory of the National Technical University of Athens, and is designed based on a specialisation of a general metadata ingestion work-flow, as it is described in detail in D7.1.

The rest of this document is structured as follows:
Chapter 3 discusses the modeling efforts in the Europeana group, including the definition and guidelines for the Europeana Semantic Elements schema as well as information on the latest Europeana Data Model and the prototyping efforts of NTUA towards adopting and interoperating with EDM. Chapter 4 describes the establishment of metadata crosswalks within ATHENA through the Ingestion Tool and discusses the achievement of interoperability within the aggregation repository. The ATHENA aggregation and ingestion services are outlined, focusing on the mapping and transformation procedure and the repository deployment. Chapter 5 presents the OAI-PMH implementation that allows for the remediation of metadata from the ATHENA ingestion platform to the Europeana repository. Finally, Chapter 6 summarizes the Conclusions of the report.

3. Europeana Metadata Modelling

3.1 Metadata Mapping & Normalisation Guidelines for the Europeana Semantic Elements

The current version, V3.42, of the Europeana Semantic Elements (ESE) is an updated version of the metadata set that has been used from the start in the Europeana prototype in November 2008. This version has been updated to take account of the Data Quality Improvement Plan which makes more elements mandatory and changes the usage of some elements. It is a Dublin Core-based application profile providing a generic set of terms that can be applied to heterogeneous materials thereby providing a baseline to allow contributors to take advantage of their existing rich descriptions. An XML Schema3 has also been produced as a further tool to assist providers in ensuring compliance with ESE.

As a Dublin Core (DC) application profile the ESE incorporates elements from the dc and dcterms namespaces plus some locally coined terms in the Europeana namespace which have been added specifically to support functionality in the portal. A full alphabetic declaration of these terms can be found in the ESE V3.4 Specification. Finally, there is also a Guidelines document4 that goes into more detail about mapping source data to the ESE format. The full set of elements is divided into those that are Mandatory, Recommended and Additional elements. There is a small section explaining the elements that contain normalised data added by Europeana. When making mapping decisions, providers are also asked to consider how their data will perform in response to "who, what, where and when" queries.

3.2 Europeana Data Model Overview

In this section a short introduction to the Europeana Data Model (EDM) is given. The interested reader is referred to the model definition5 and primer6 documents for a complete and comprehensive reference to EDM. EDM was proposed in order to structure the data that Europeana ingests, manages and publishes and it is a major improvement of Europeana Structural Elements (ESE), which was the initial data model that Europeana began life with. Figure 3.1 graphically summarizes the hierarchy of the EDM classes.

[2] http://www.europeana-libraries.eu/c/document_library/get_file?uuid=77376831-67cf-4cff-a7a2-7718388eec1d&groupId=10128

[3] http://www.europeana.eu/schemas/ese/ESE-V3.4.xsd

[4] http://www.europeana-libraries.eu/c/document_library/get_file?uuid=b3cfcf47-da0a-4c6b-b1d7-9b08e162643e&groupId=10128

[5] http://group.europeana.eu/c/document_library/get_file?uuid=aff89c92-b6ff-4373-a279-fc47b9af3af2&groupId=10605

[6] http://group.europeana.eu/c/document_library/get_file?uuid=718a3828-6468-4e94-a9e7-7945c55eec65&groupId=10605

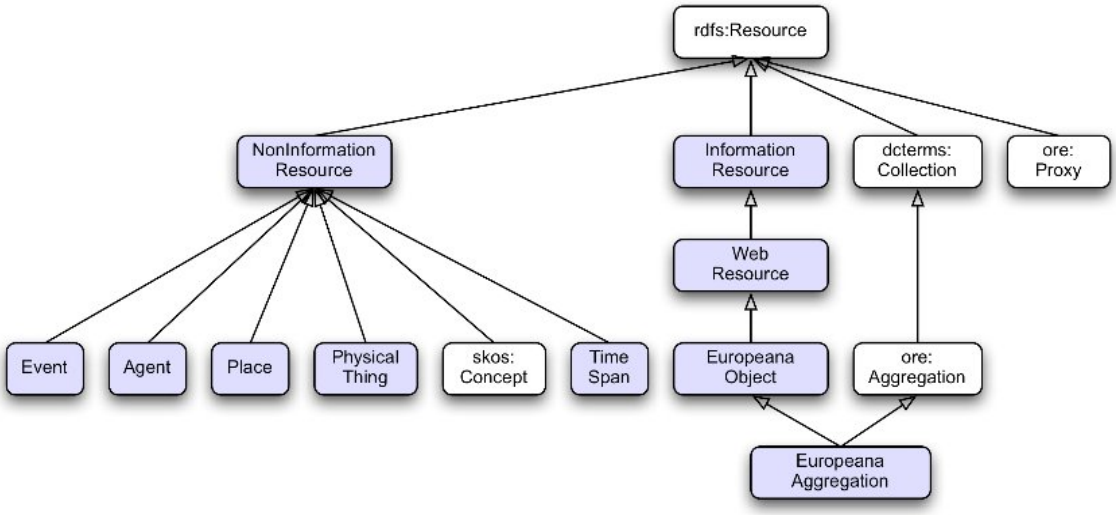

Figure 0.1 The EDM Class hierarchy. The classes introduced by EDM are shown in light blue rectangles. The classes in the white rectangles are re-used from other schemas; the schema is indicated before the colon.

As we can observe, the classes that are defined in EDM (blue rectangles) can be divided into NonInformation and Information Resources. Information Resource class represents resources whose essential characteristics can be conveyed in a single message e.g. a text is an Information Resource. Web Resource is a subclass of Information Resource and more specifically it is defined as an Information Resource that has at least one Web Representation and at least a URI. EuropeanaObject and EuropeanaAggregation are subclasses of WebResource, but since these classes are only used from Europeana for managing the data and they are beyond the interest of this document.

On the other hand, all the resources that are not Information Resources are instances of the NonInformation Resource class that has various subclasses. Firstly, class Event represents a change of states in cultural, social or physical systems, regardless of scale, brought about by a series or group of coherent physical, cultural, technological or legal phenomena or a set of coherent phenomena or cultural manifestations bounded in time and space. The second subclass of NonInformation Resource, that is Agent class, comprises people, either individually or in groups, who have the potential to perform intentional actions for which they can be held responsible. Class Place represents an extent in space, in particular on the surface of the earth, in the pure sense of physics: independent from temporal phenomena and matter. The next one is PhysicalThing class which represents a persistent physical item such as a painting, a building, a book or a stone. Finally, class TimeSpan is an abstract temporal extent having a beginning, an end and duration.
The following figure illustrates the properties defined in EDM to interconnect the classes. Detailed explanation of their semantics is not given in this section. A short description for some of the properties is provided in the next section.

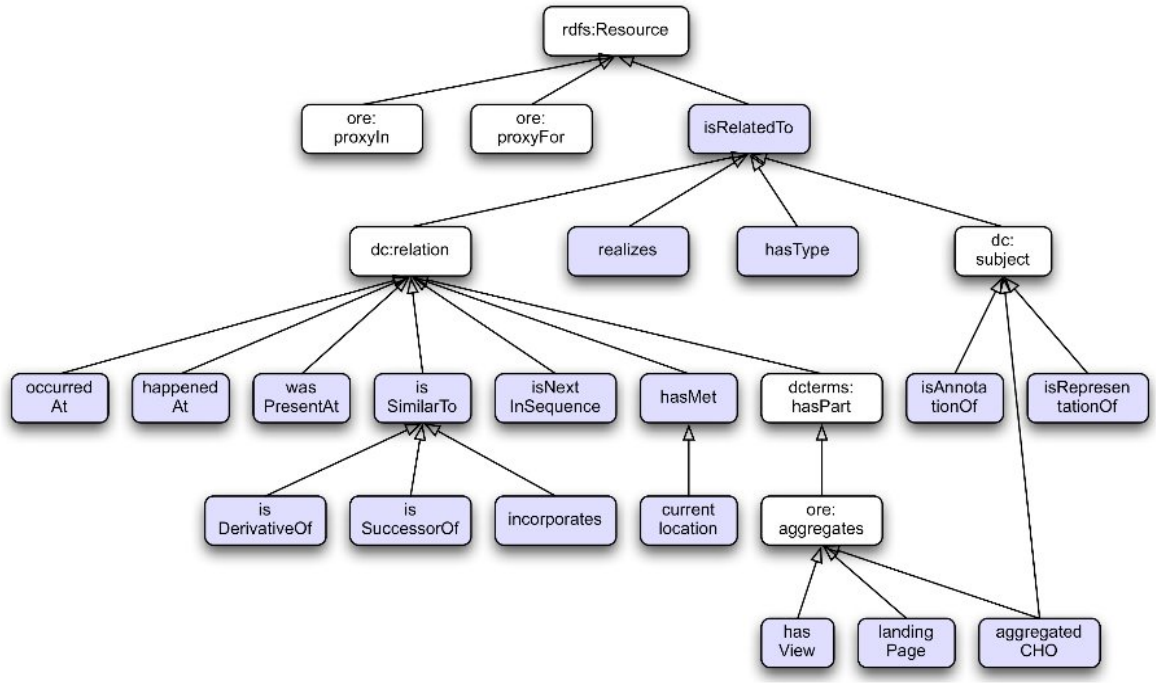

Figure 0.2 The EDM property hierarchy without the properties included in ESE (for readability). The properties introduced by EDM are shown is light blue rectangles. The properties in the white rectangles are re-used from other schemas.

The overall data structuring within EDM is based on two classes borrowed from Object Reuse and Exchange (ORE) schema ore:aggregation and ore:proxy. Every cultural heritage object (e.g. a painting) has a digital representation (e.g. a thumbnail of painting's digital picture). Therefore a distinction between the actual works and the digital representation is necessary. EDM manages this distinction by following the ORE specification and using aggregations. Hence, EDM considers that the provided object, together with the digital representations that are contributed by one provider, form an aggregation. Proxies, on the other hand, are used in order to handle the fact that Europeana takes data from many providers and this data may be about the same real world resource, thus giving multiple views on the same resource. In addition, Europeana can add its own data about that resource giving yet another view on the same resource. Therefore, since it very likely that the metadata differ, e.g., different names may be used for the same creator, these views remain distinct by using ore: proxies.

To better understand the way data are represented using EDM, let us assume that we have two records of Mona Lisa, one from the Joconde database and another from the Louvre. Each data submission to Europeana gives rise to a specific instance of the ore:Aggregation class, used to group all the elements related to one resource that come from one provider. Both providers indeed contribute a different set of digital representations, e.g., different resolutions, different file types and, of course, different locations for the representations. In this way an aggregation is one provider's contribution for an object. Moreover each metadata record provided to Europeana also gives raise to a specific proxy for the object described, modelled using the ore:Proxy resource. This proxy is specific to a given provider, and is used to represent the description of the provided object, as seen from the perspective of that specific provider. With proxies it is possible to represent different, possibly conflicting pieces information on provided objects, while still keeping track of the provenance of this information. A proxy is connected to the resource by using the ore:proxyFor

property while it is connected to its provider's aggregation using the ore:proxyIn property. Figure 0.3 provides a graphical representation of the described example.

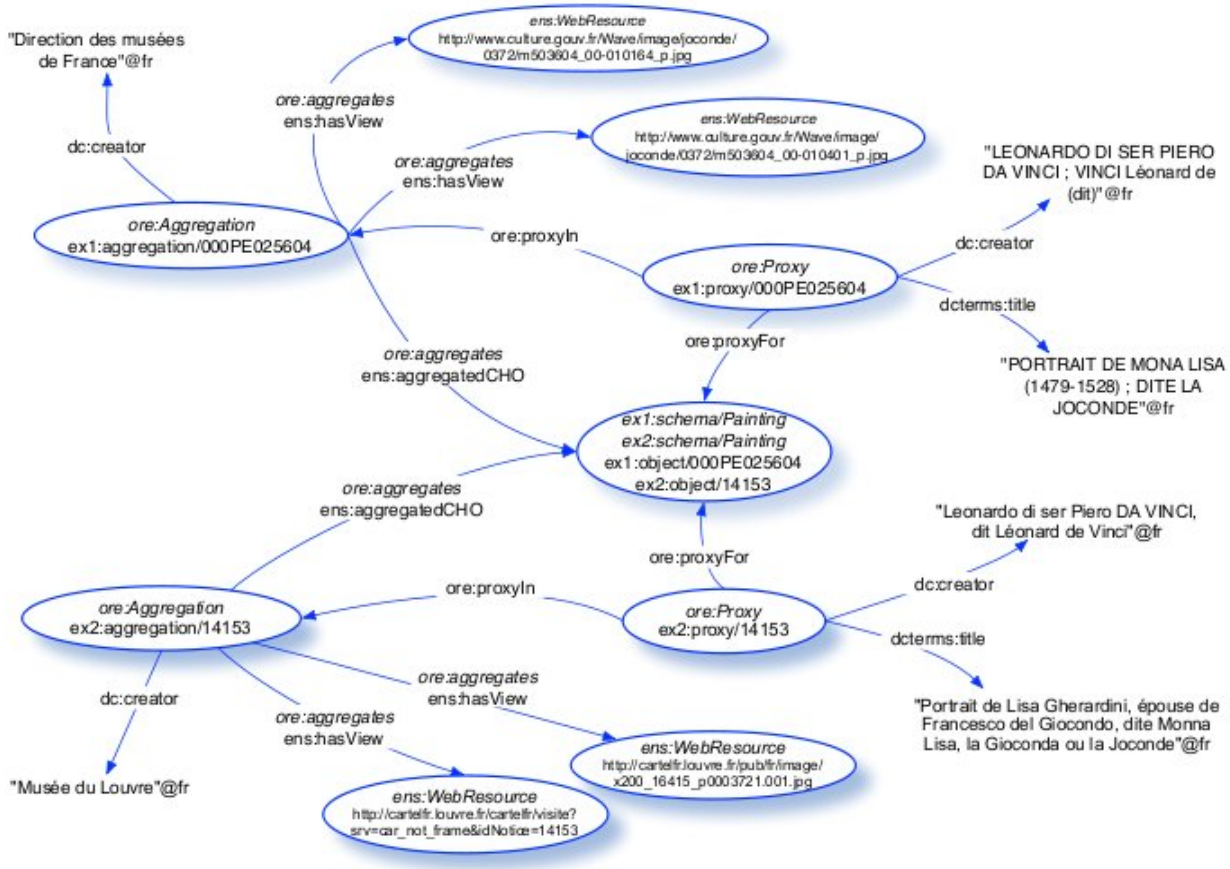

Figure 0.3 EDM Representation for two providers for an object.

3.3 EDM Harvesting XSD

An xml schema was implemented based on the EDM in order to mediate between content provider's native data and the data models implemented by aggregators. More specifically the main requirement for this schema was to act as a harvesting XML schema intended for delivering metadata to the aggregation service environment about an organisation's online collections and digital objects. Furthermore, this schema had to ensure interoperability between the native metadata held by heritage organisations and the standards and schemas used in Europeana.

Having in mind the EDM, we implemented a schema that is based on EDM's classes. In general the design methodology that was followed for the implementation of EDM XSD was to construct complex types for the EDM classes that are comprised of elements that correspond to the EDM properties. Therefore, since according to EDM for every ingested item given by the provider an aggregation is constructed, we defined a complex type named AggregationType to represent aggregation..

As illustrated in Figure 0.4 AggregationType consists of 4 mandatory elements which are the proxy, the aggregated Cultural Heritage Object (CHO), the webResources and the creator. Among them only the creator is of SimpleLiteral type, while the remaining are complex types in order to represent the corresponding EDM classes.

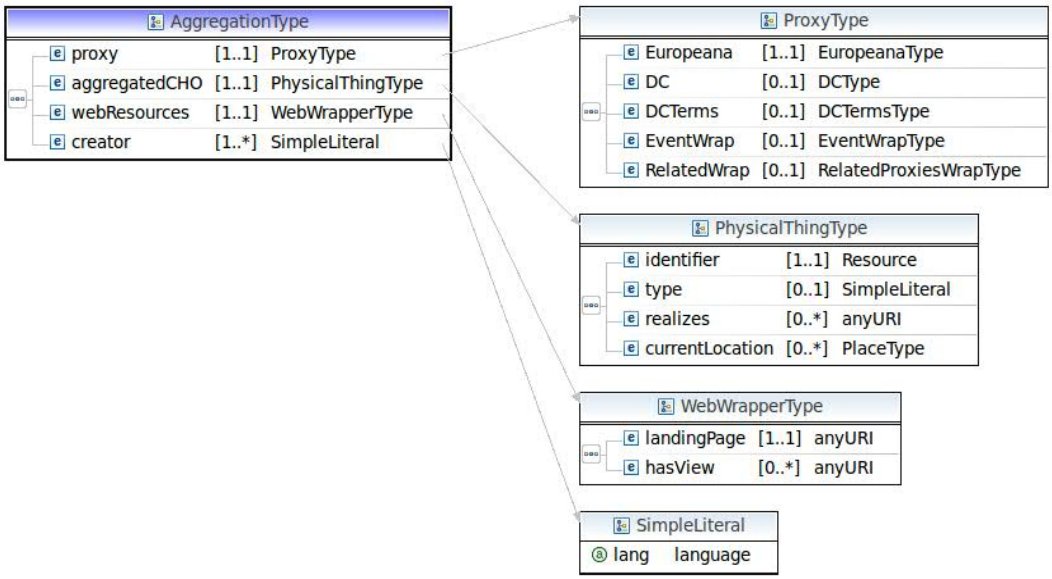

Figure 0.4 Overview of EDM XSD

ProxyType is a complex type repsesenting EDM's class proxy. Since proxy class is responsible for holding all the metadata of the CHO, it consists of five other complex types. DC and DCTerms elements, which are illustrated in the following figures, were generated to collect the metadata borrowed from the DC and DCTerms schemas respectively.

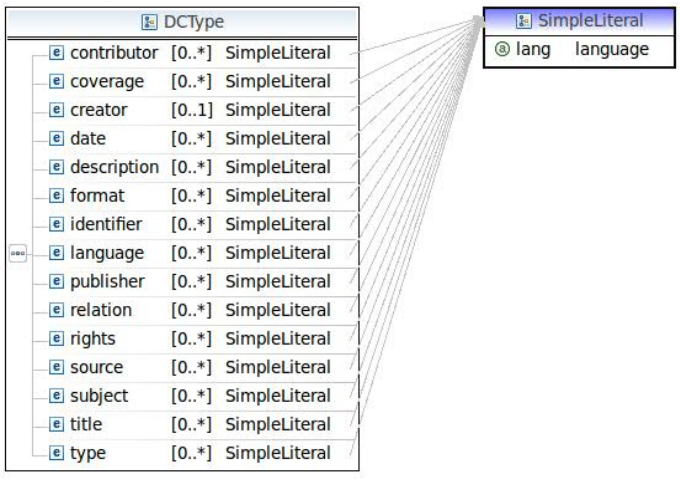

Figure 0.5 DCType

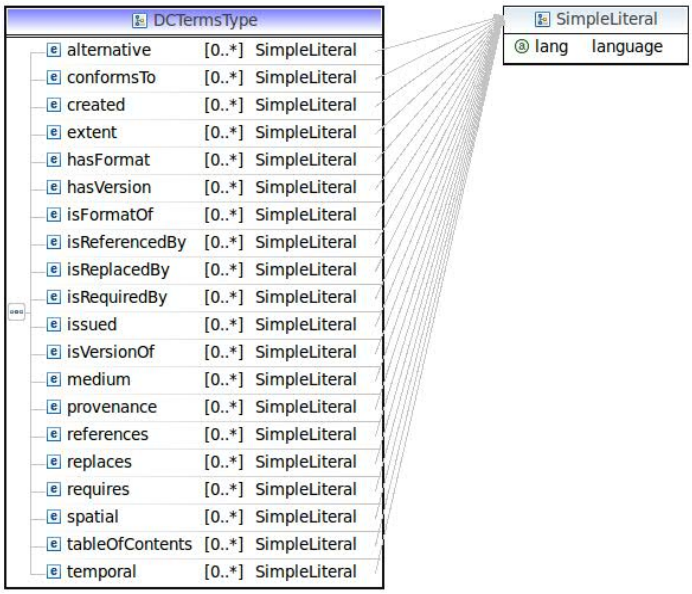

Figure 0.6 DCTermsType

None of the elements included is mandatory and their type is SimpleLiteral. A more correct representation, in terms of DC and DCTerms properties semantics, would be to set the type of these elements to SimpleLiteral or anyURI, since these properties are defined as RDF properties in DC and DCTerms respectively. However, this representation would prove rather confusing in our case, since the scope of EDM schema is to act as a harvesting schema and the existing metadata about CHOs are literals and not resources.

The third complex type that is included in ProxyType is the EuropeanaType that is the only mandatory element in proxy and includes the properties defined in EDM (see Figure 0.7). In this case, there exist mandatory elements, and not all elements are of the same type. The mandatory elements were defined according to the EDM. Hence the properties that are specified with cardinality 1 were converted to the mandatory elements. These are the country and the language that are of SimpleLiteral type, the provider that is of string type, since different languages cannot apply to this field, and the type that is of Edmtype type. Edmtype is a string which can only take one value from TEXT, VIDEO, IMAGE and SOUND defined in that way in accordance to the EDM specification for the type property. For the same reason (i.e. because of the EDM specification) the type of rights and uri elements was set to anyURI.

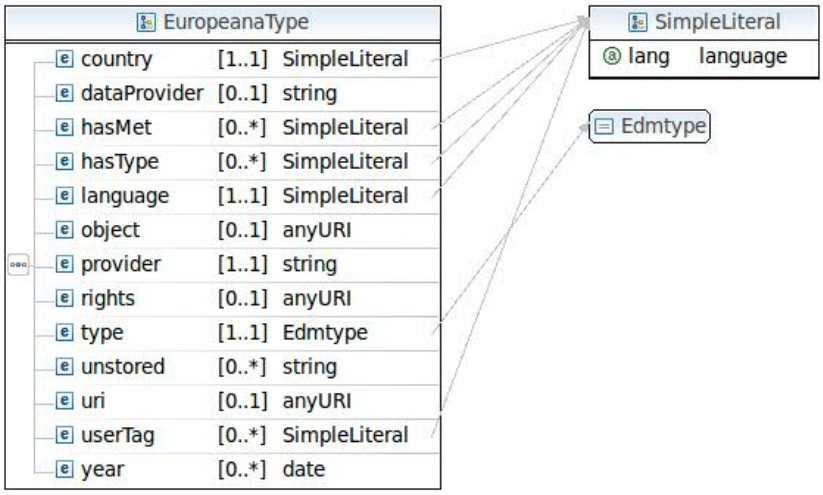

Figure 0.7 EuropeanaType

The fourth element that is included in ProxyType is the EventWrapType, which was implemented to in order to fit the EDM class Event. More specifically this type represents the EDM property wasPresentAt which
relates a Proxy with an Event. At this point, we should mention that EDM permits both “object centric” and “event centric” approaches for metadata. The first one focus on the object described, in other words information comes in the form of statements that provide a direct linking between the described object and its features. Object centric approach is represented in our schema by DCType and DCTermsType. Event-centric approach, on the other hand, considers that descriptions of objects should focus on characterizing the various events in which objects have been involved. This approach underlies models such CIDOC-CRM and is represented in our schema by EventWrapType and EventType.

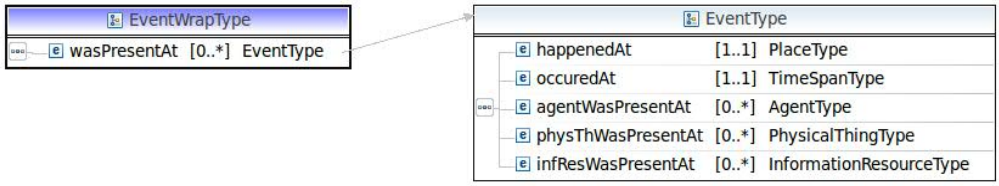

Figure 0.8 EventWrapType

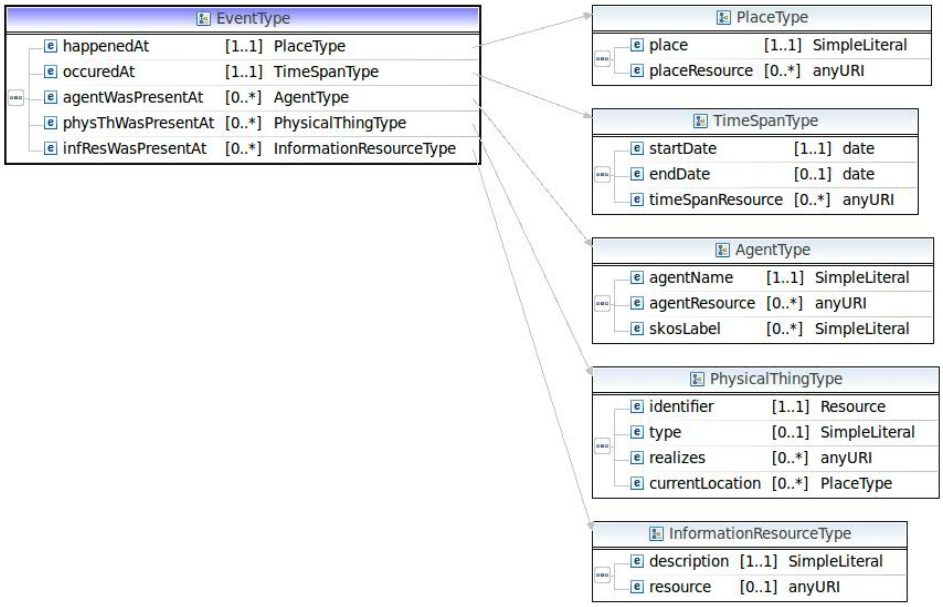

Figure 0.9 EventType

EventType includes elements happenedAt and occuredAt that are mandatory for an event and are of PlaceType and TimeSpanType types respectively. Having a Linked on Data (LOD) representation in the back of our minds - especially for the event centric approach - we added elements that are not specified in EDM. Therefore a place is represented by a simple literal while a resource for it, if available, is optional. In a similar manner TimeSpan is represented by the mandatory element start date and the optional end date and resource.

The remaining three optional elements that are included in EventType are special conditions of the wasPresentAt property. More specifically the domain of property wasPresentAt is defined in EDM as the union of class Agent, InformationResource and PhysicalThing. In other words any of them can be present in an event. Therefore each of AgentWasPresentAt, PhThWasPresentAt and InfResWasPresentAt correspond to Agent, InformationResource or PhysicalThing complex types that indicate their presence in an Event. As before, these complex types are represented with a mandatory field, agentName identifier and description repsectivelly, together with other optional elements among them a resource. It is important to mention at this point that since it would be quite rare to have the resource for these complex types (Agent, Place, TimeSpan, InformationResource, PhysicalThing) and always having in mind a LOD representation, some additional elements may be added in the schema for them. In that way there will be more information about them and therefore a resource discovery operation by searching is specific datasets would be feasible.

The last element that is included in the proxyType is the relatedProxiesType. This compex type was implemented to represent properties defined in EDM that connect two different proxies. Therefore the element RelationType is a string enumeration of these properties that are hasPart, isDerivativeOf, isSuccessorOf, incorporates, isRelatedTo and isNextInSequence while the proxyUri specifies the target-object proxy through its URI.

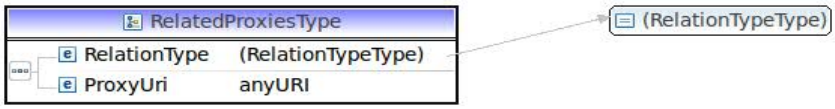

Figure 0.10 RelatedProxiesType

Finally, going back to AggregationType (Figure 0.4) from where we started, the two remaining elements are arggregatedCHO that is of PhysicalThingType and webResources of WebWrapperType. AggregatedCHO represents the CHO that is described while the webResources element collects the webResources for the CHO that are the landingpage i.e. the provider's web page about the CHO and the shownAt page that is a web page with the thumbnail of the CHO.

3.4 EDM RDF

The purpose of EDM XSD is to collect metadata about CHOs for delivering them to the aggregation environment about an organisation's online collections and digital objects. Since it is based on the EDM that is ontology, an XML instance of this XSD can be converted to an RDF instance based on EDM. More specifically as mentioned in the previous section the design methodology, based on which XSD implemented is to create complex types for the classes that include elements which are the properties of EDM. Hence, RDF construction is done by creating an Aggregation resource according to the metadata entered in AggregationType that specify the RDF interconnections. A graphical example of the RDF that is created for MonaLisa from an instance of EDM XSD is illustrated in Figure 0.11.

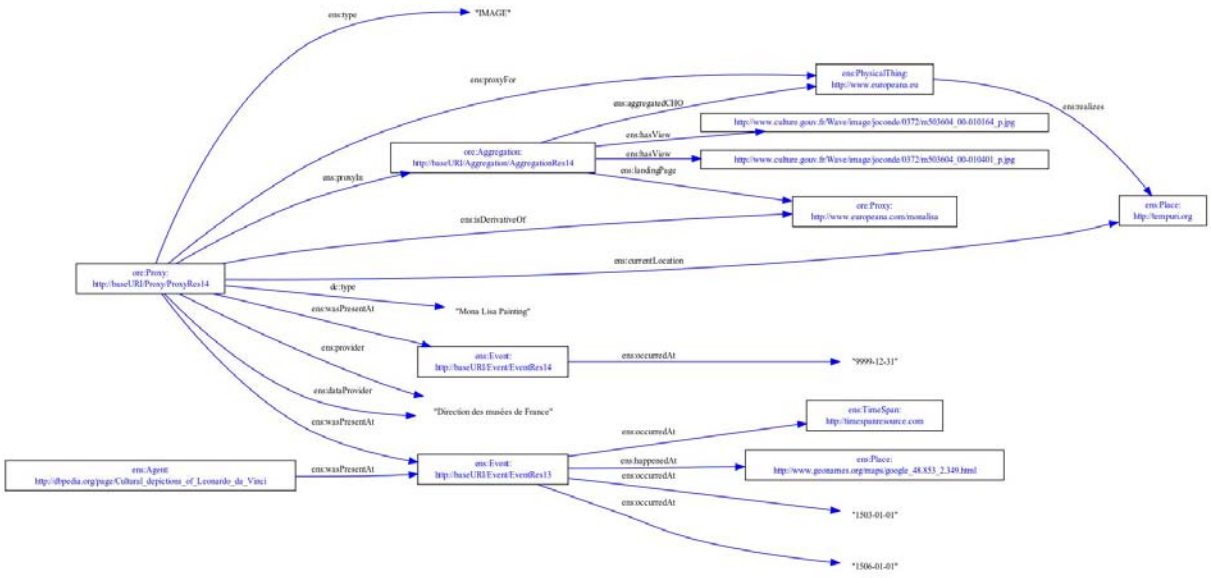

Figure 0.11 EDM RDF The Mona Lisa example.

4. Metadata Crosswalks and Interoperability in ATHENA

4.1 ATHENA Metadata Aggregation Platform

WP7 has setup a platform to offer services for content providers in order to perform ingestion and aggregation of metadata, leveraging the expertise and available resources regarding metadata crosswalks. Due to the nature of a European thematic aggregation there is an expected diversity among participating providers and content. The software tools need to accommodate inexperienced users and legacy data, while taking advantage of the domain experts' knowledge and the project's working groups' results. A tool for visually mapping local metadata schemas to the adopted ATHENA reference model, LIDO7, is one that will ensure the success of a large scale aggregation by providing an intuitive, user-friendly approach that reduces the effort to create translation logic for mappings and the turnaround time between human-readable crosswalks and executable code.

Content upload

Currently, allowed metadata formats for uploads are:

- XML in any schema.
- zip archives of the above

The following methods are supported for uploading content:

- HTTP upload; suggested only for relatively small amounts of data (<2MB)
- Upload to a dedicated FTP server.
- Remote HTTP or FTP browsing.
- OAI-PMH repository harvesting.
- SuperUser uploading from local file system (restricted).

User and organisation management

Users belong exclusively to one organization and cannot access data non related to that. Users can be assigned with different levels of access, that grant roles ranging from data browsing, over editing and annotating, to being allowed to edit other users' details (administrators). We have extended user roles to allow parent users, for organizations that might not have expertise or manpower to use the system and thus, delegate the job to an organization which is then their designated "parent" organization. Parent users extend their rights to child organizations and provide the functionality to build the access hierarchy for any given country/thematic category. The current role set can be easily adjusted to allow more freedom in rights management.
The following rights are currently implemented for metadata:

- change/add/delete user
- change/add/delete organization
- edit/upload/delete metadata
- publish / declare finished metadata sets
- read-only browsing rights

These rights have been grouped to the following roles:

- Administrators (all user, data and organization rights for the organizations they manage),
- Annotators (data management rights),

[7] http://www.lido-schema.org/

- Publishers (publishing data rights),
- Data Viewers (simple viewing rights).
- The system also contains some hardcoded super-users that have full rights to all organizations and their data in the system.

Functionality and user interfaces

Users can join the ATHENA metadata ingestion service using the registration page (Figure). During registration they are prompted to select the organization they belong to.

Figure 4.1 Registration Screen of ATHENA Metadata Ingestion Service

The administrator of that organization is notified by email for the pending user registration and is authorized to grant the appropriate rights and finalize the procedure. In case a user's organization is not present in the list of registered organizations, the user can register in the system without providing one. In this scenario he is given the opportunity to create a new organization and automatically become the administrator for it.

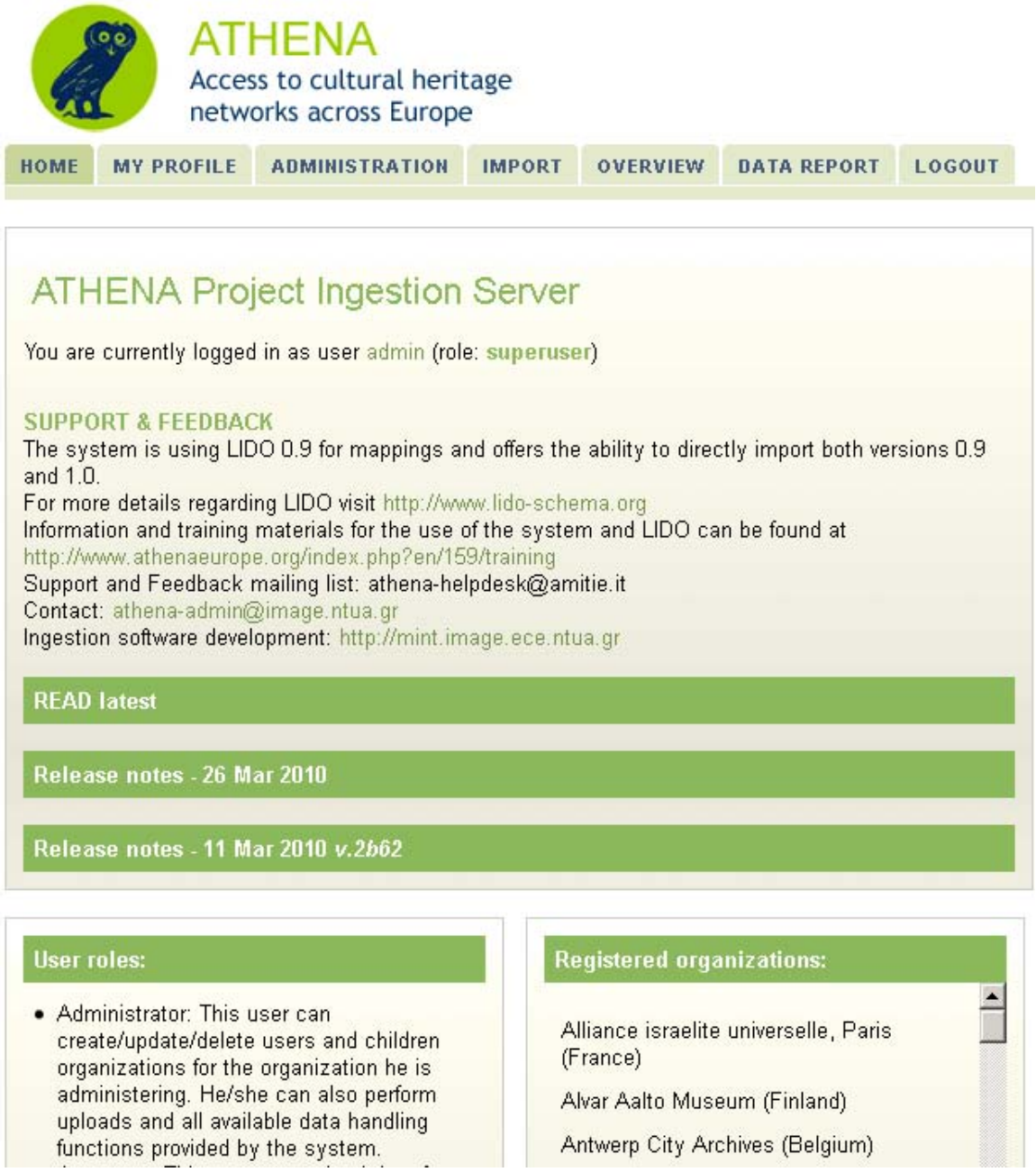

Figure 4.2 ATHENA Metadata Ingestion Service - Home screen.

Organizations within the system can have parental organizations. Users of parental organizations extend their rights to the children of the organization (and in turn to grandchildren that may exist, and so on). Every organization can have at most one parent organization. The parent organization has to agree on publishing the data of the child organizations (among other things). This way an aggregator for example can define and manage all the organizations (and their respective data) he is

supervising within ATHENA.

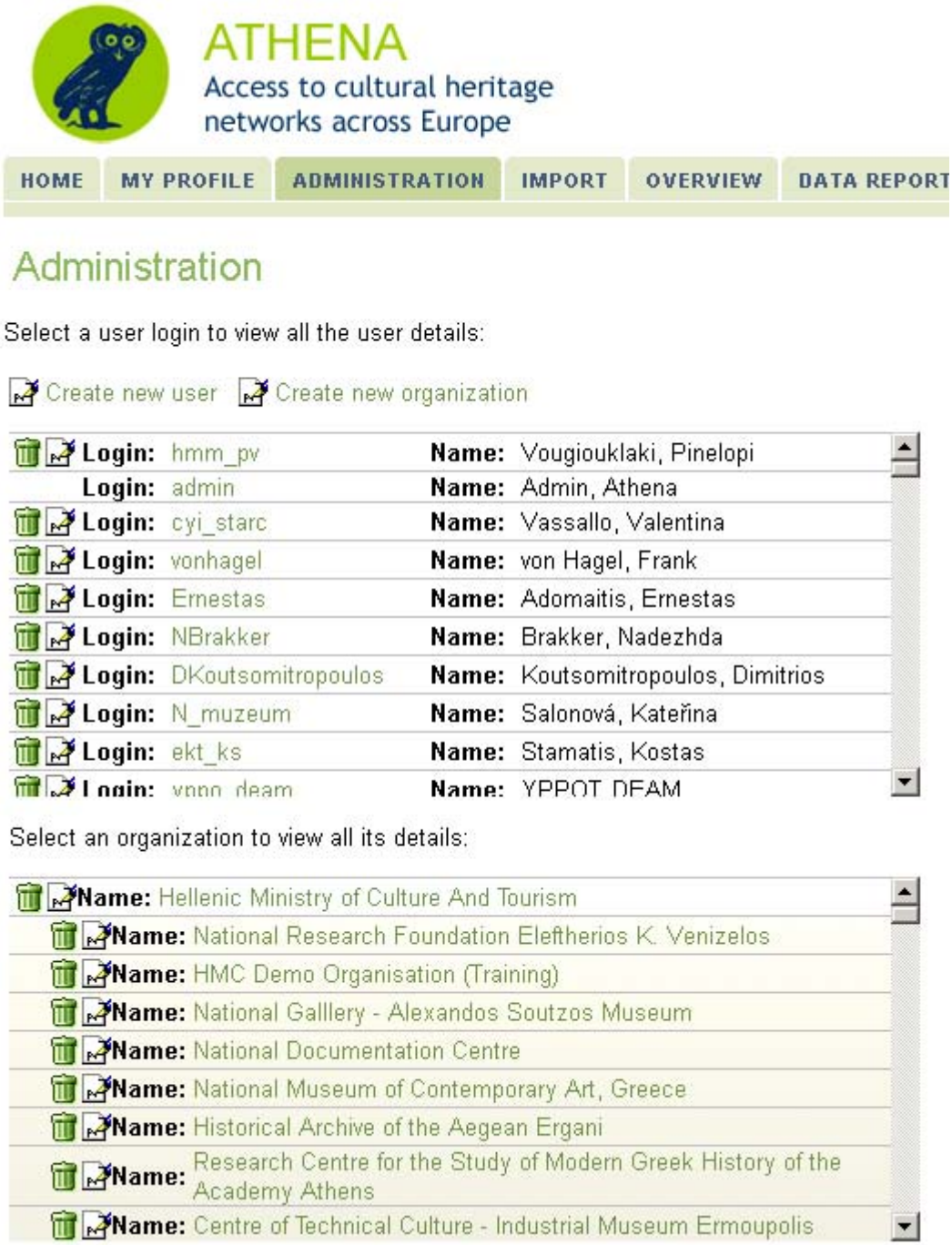

Figure 4.3 ATHENA Metadata Ingestion Service - User's Administration Screen.

Every user of the system has the right to see all the other users and data within the same organization (Figure). S/he can change her own details by using the Profile page (Figure).

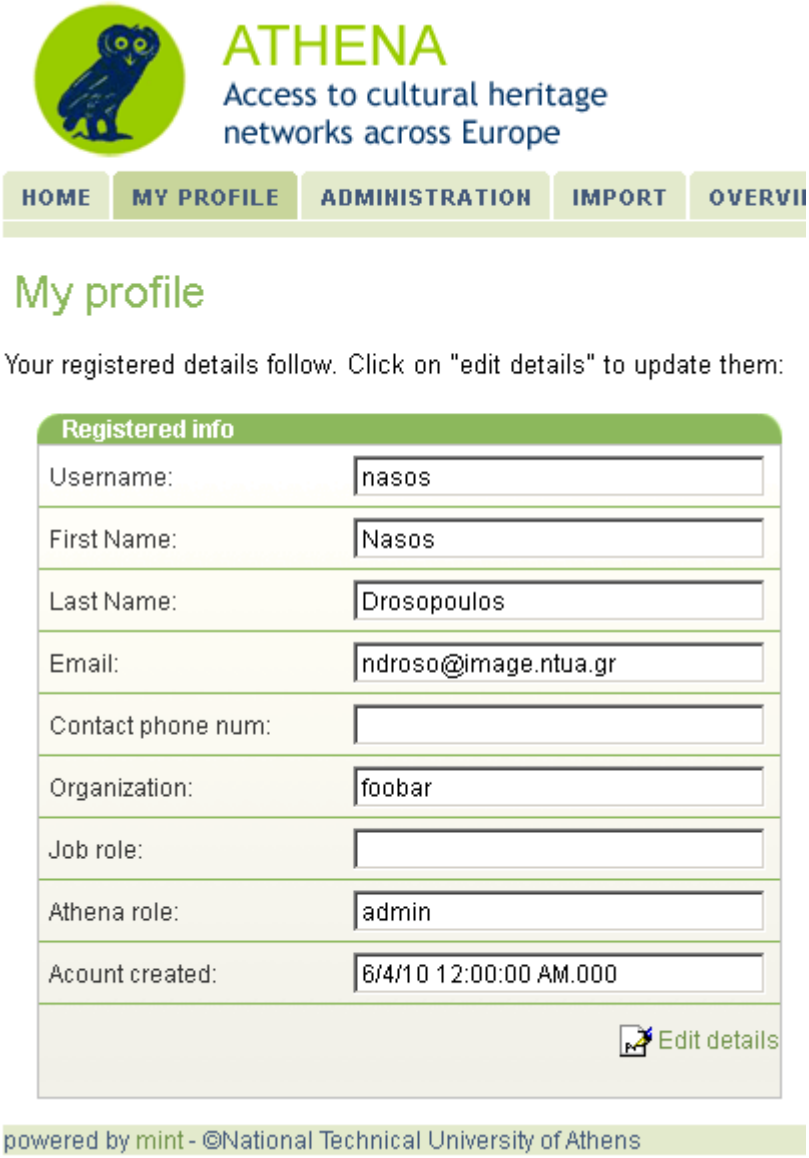

Figure 4.4 ATHENA Metadata Ingestion Service - User Profile Screen.

Administrators and Annotators of an organization can upload data using the import page (Figure).

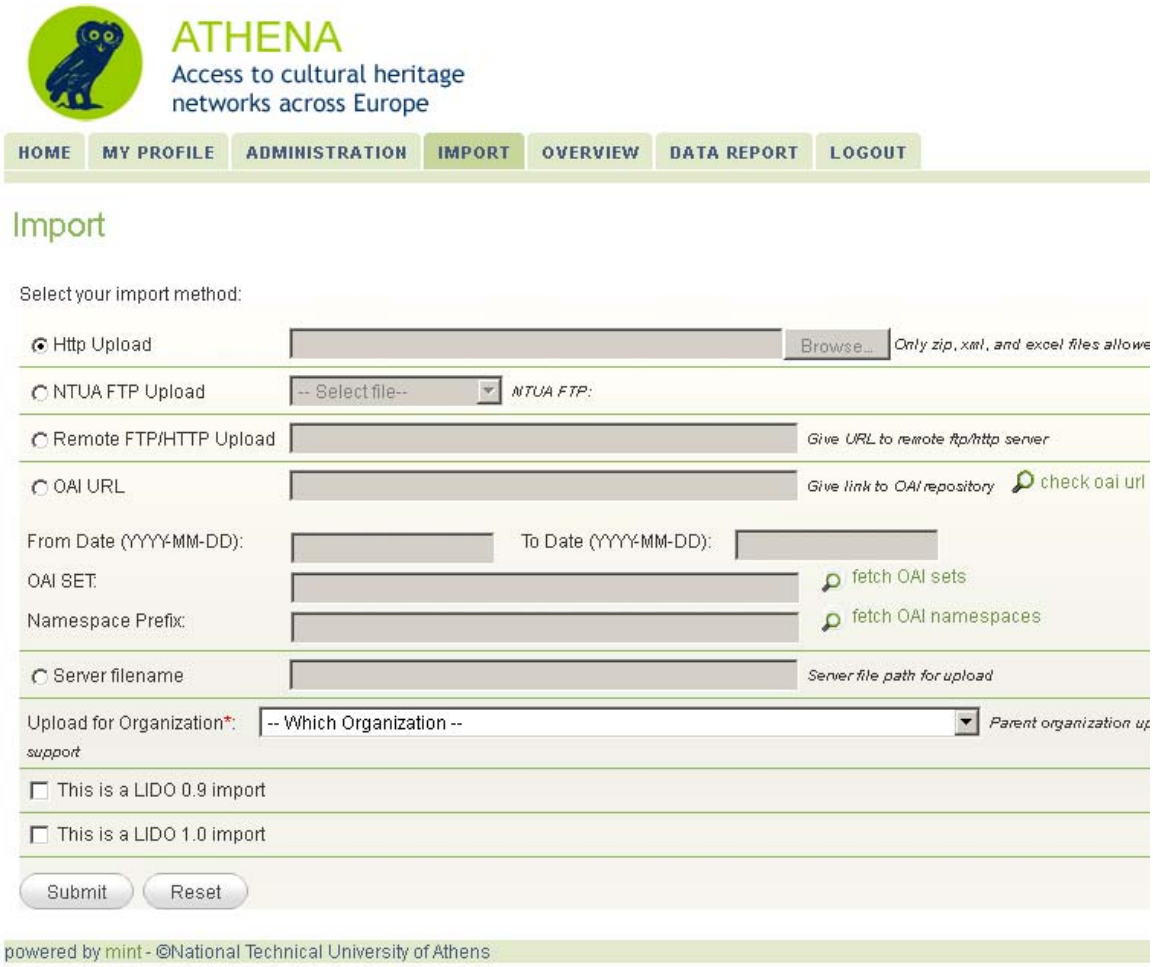

Figure 4.5 ATHENA Metadata Ingestion Service - Import Interface.

A history of all uploads for an organization can be browsed using the Overview interface Figure). Different icons are used to show the current status of an import (hourglass for processing, green arrow for completed, red 'x' for failed).
An import can be deleted, thus deleting all items it contains. After an import process has been completed, mappings have to be defined for the data set in order to see all the items it contains. Every mapping defined for an organization can be saved, edited and reused at a later stage.

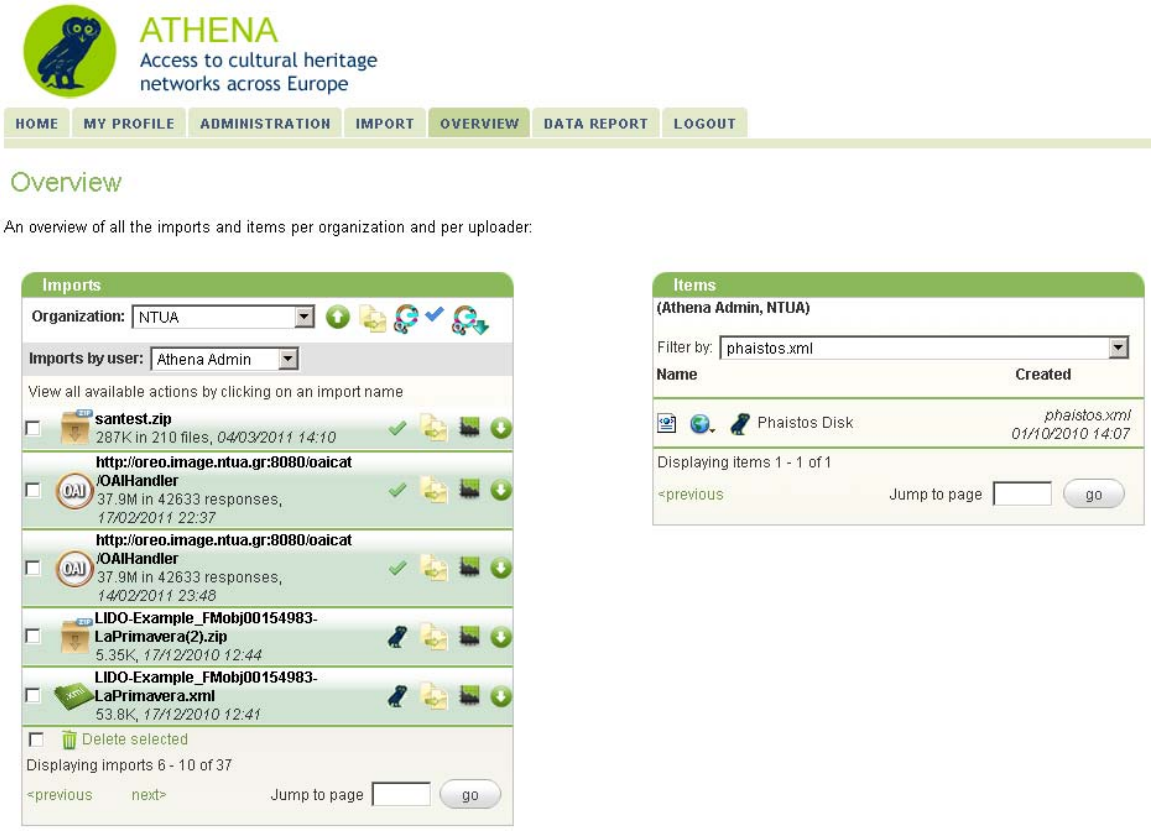

Figure 4.6 ATHENA Metadata Ingestion Service - Overview Interface.

In the overview screen the user can browse through items that have been uploaded. Metadata can be uploaded in a single XML containing multiple items or in multiple XML files, each one containing one item. In order to preview the items that belong to each upload, the user has to firstly define the item‘s root element in the XML structure (Figure) and additionally an element that will serve to label the separated items. The actions that are available to the user as part of the Overview Interface, as presented in Figure , are the following:

- When a “Green” tick appears, near the import name, it indicates that the importing process was successful. If a “Red” cross appears it means that the importing process has failed. In any case if the user hovers the mouse over the icon; various information regarding the process is displayed.
- . This is the Show Items icon. When the user presses this button a new modal window is entered where he/she is able to review the dataset of the import. More details about this functionality in the “Review Original Dataset” chapter.
- . This is the statistics button. When the user presses it a new modal window is rendered, where information regarding the imported dataset is presented. All the different XPaths that were extracted are presented in a tabular form together with statistical information regarding the distribution of the various values of each XPath.
- . The download button. When the user presses it he/she is able to download an archive with all the XML files that are part of the ingested dataset.

Figure 4.7 *How the imports are presented to the user in the “Overview” Tab.*

When the user clicks on the name of the Import an extended view for the current Import is presented as depicted in Figure . The rest of the mandatory steps that are part of the ingestion tool core workflow are executed from this extended view. More specifically:

-  . This button invokes the process for defining the root and label element from the extracted XPaths from the ingested data set.
-  . This button executes the transformation of the items.
-  . This button invokes the mapping tool in order to perform the semantic mappings between the source and target Schemas and produce an XSLT.

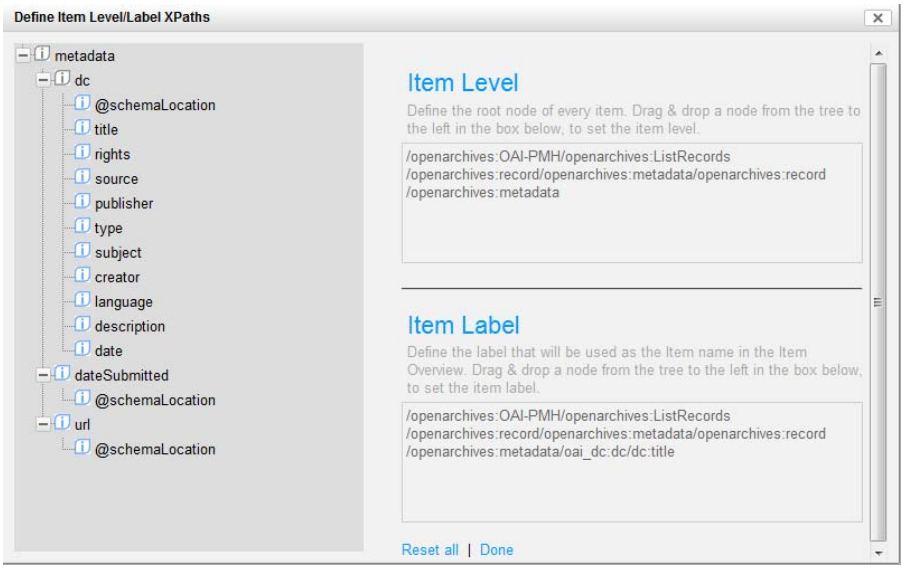

Figure 4.8 *ATHENA Metadata Ingestion Service - Definition of item root element.*

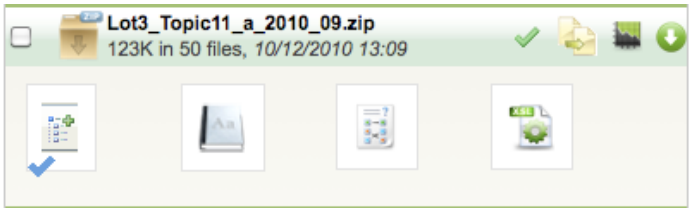

Figure 4.9 *The extended view of an Import in the "Overview" Tab.*

Having defined the item root element, the items that belong to the specific upload can now be previewed in Figure . Finally, through an icon next to each item, the user views the original XML (Figure)

Figure 4.10 *ATHENA Metadata Ingestion Service - Items Screen.*

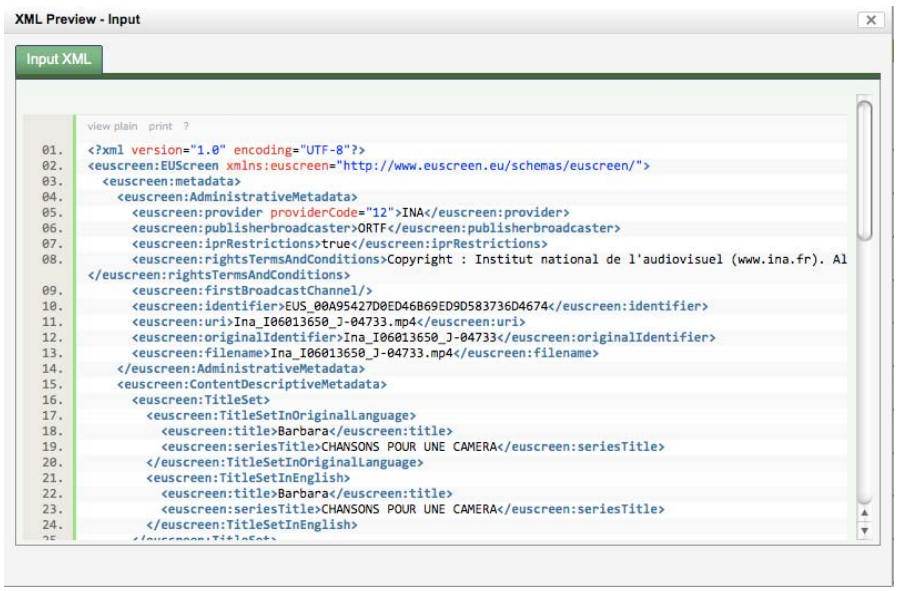

Figure 4.11 *ATHENA Metadata Ingestion Service - Input XML.*

4.2 The Mapping Module

The core module of the ATHENA Metadata Ingestion tool is the mapping tool. Although the service shares functionality with many existing metadata repositories, e.g. DSpace and Fedora, one of its main goals it to provide support for a great diversity of metadata schemas or simple data structures, thus widening metadata interoperability. The ATHENA Metadata Ingestion platform aims to be able to store and manipulate metadata that are described using different conceptual models for encoding and decoding information. For this reason both a syntax and semantics have to be defined in order to obtain a complete and expressive model. XML is used as the machine understandable syntax and can be interpreted using different parsers depending on the specified needs. The mapping tool provides the interfaces and mechanisms for identifying and registering through a reference model the semantics of the models used.

Data integration processes comprise of various tasks including data matching, data transformation, and schema/semantic matching. Many solutions have been proposed by the community for each one of those tasks, ranging from applications that rely heavily to the user, to applications that are semi-automatic and in some cases completely automatic depending on the task, thematic category and schema complexity. For the case of schema/semantic matching many techniques and platforms have been developed, enabling the user to complete successfully the task. Notable cases are the schema mapping tool provided by Altova that offers a rich editing environment where the user is able to map any number of arbitrary schemas, but the whole process is totally manual, and the COMA++ platform that offers the user an environment for semi-automatic schema mapping, using state of the art algorithms. Although these approaches attempt to solve the general problem, the case of the ATHENA aggregation has specific characteristics that lead to a more specialized solution to efficiently handle large amounts of diverse data and metadata.

By choosing a semantically rich and well-defined reference schema as the target of the mapping process the user has the opportunity to semantically enrich his data and metadata while quality of the aggregated content is ensured. The mapping process is manual and the tool offers previewing, assisting and validation capabilities in order to ensure the quality of the result. The design principles of the mapping tool ensure the extensibility of the tool itself on a software level and of the system on a data level.

Mapping Editor

The metadata mapping tool allows the user to define semantic mappings between the source and target schemas. An XSLT is then generated, based on these mappings that can automatically convert all imported items. An example of the mapping tool is depicted in Figure .

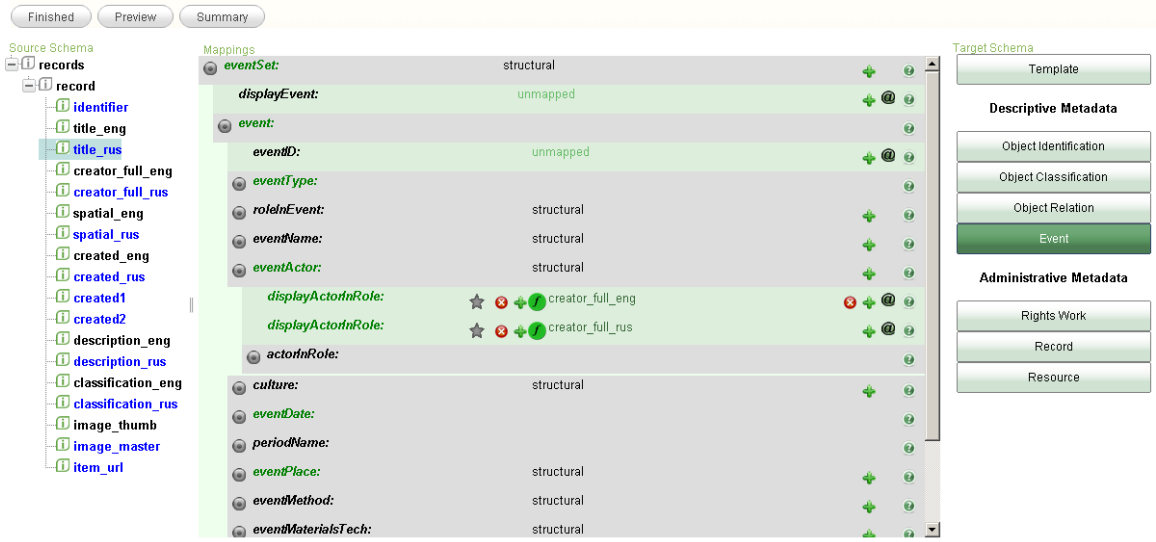

Figure 4.12 *ATHENA Metadata Ingestion Service - Mapping tool.*

Source schema

On the left of the mapping tool is a tree like structure of the source schema similar to the one presented during dataset item level selection. The user can navigate in the schema by clicking the nodes on the left of the tree elements. Elements with a grey information icon (ⓘ) are structural elements and do not contain data values. The rest of the elements are leaf elements with data while elements starting with '@' are attributes of the corresponding father. Elements in blue are elements that have been already used in this mapping. More information about the schema elements is provided by clicking the ⓘ icon. This action invokes a panel as show in the following Figure:

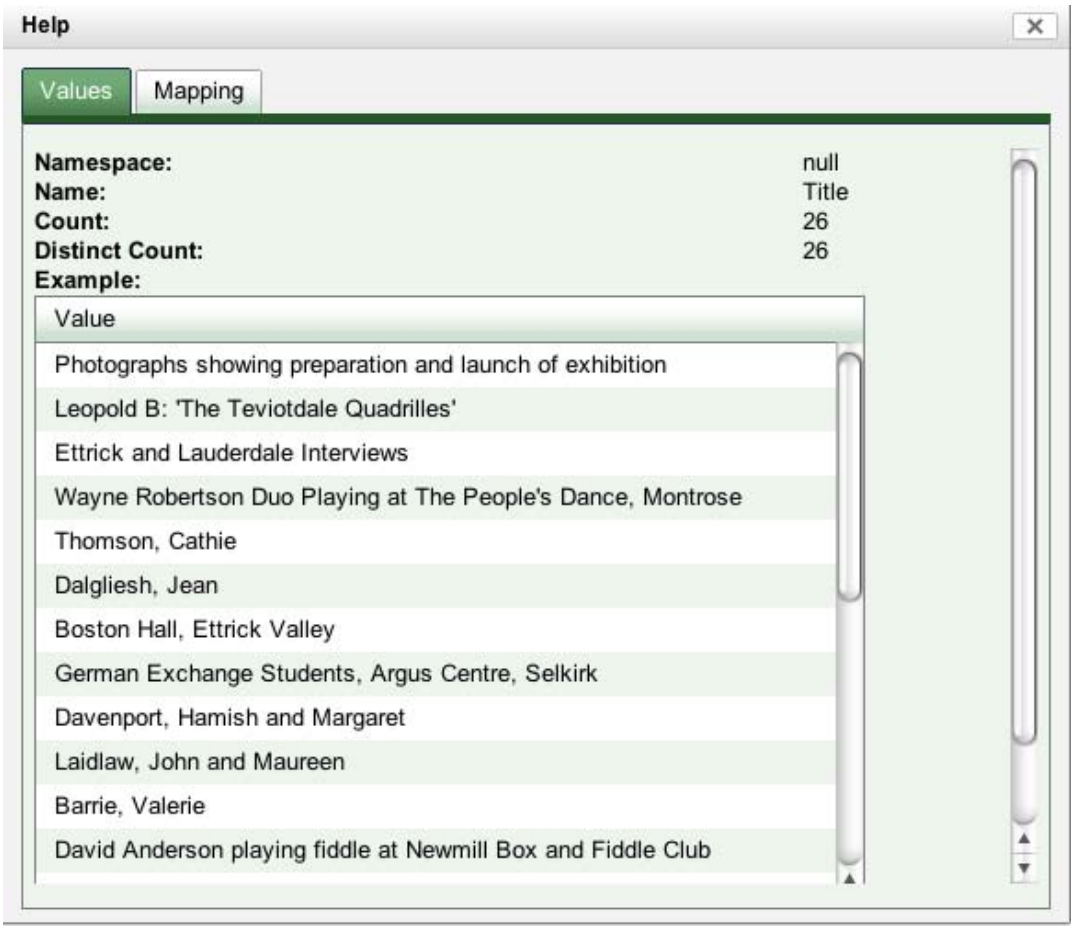

Figure 4.13 *ATHENA Metadata Ingestion Service - Source schema element information panel.*

This panel contains two tabs: Values and Mapping. The Values tab shows the following information about the specific element:

- **Namespace:** The XML namespace to which this element belongs.
- **Name:** The element name.
- **Count:** The number of times the XPath of this element exists in the imported dataset.
- **Distinct Count:** The number of unique values associated with this XPath in the imported dataset.
- **Example:** A sample of these values sorted by their frequency of appearance in the imported dataset.

The Mapping tab shows where and how the specified element is being used in the mapping that is being edited.

Target Schema & Mapping Area

The target XML schema is split into sections that appear as buttons on the right side of the mapping tool. These buttons are used to navigate to specific parts of the target XML schema. By clicking these buttons, the corresponding part is loaded in the middle of the mapping tool along with its specified mappings.

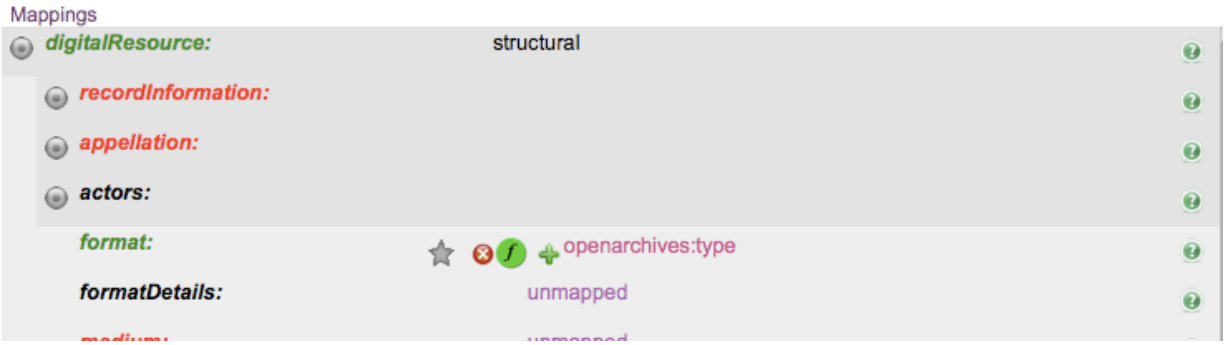

Figure 4.14 *Example of the mapping definitions.*

Each row in the mapping area corresponds to a mapping element of the target xml schema. Rows with grey background are structural elements and contain other elements. They can be expanded and their can be seen by clicking the button to left of their name. The rest of the elements can contain data and have an 'unmapped' area which can be used to defined mappings, as explained later. Elements can also have attributes which can be seen by clicking the icon when available. If an element can exist more than one times in the target XML file then a icon is available on the right of the element row. By clicking this icon an additional row will appear on the mapping area. More information about each element can be provided by clicking the corresponding icon on the right of the element row. Elements with green names are elements with mappings or have children with mappings. Elements with red names are mandatory elements that have no defined mappings or have mandatory children with no mappings defined.

Mapping types

Various mapping types are available by using the mapping tool. The most common type of mapping is **XPath mapping**. This type of mapping will copy data from a source element to a target element. To define this kind of mapping, drag n' drop a source element to the 'unmapped' area of a target element. The source element will then appear in the unmapped area of the target element. By clicking the icon on the left of the source element name, an additional unmapped area will appear for the same target element. More mappings can be performed on this unmapped area and the XML result will contain a concatenation of the provided values. Clicking the corresponding icon will remove a defined mapping.

Double clicking on the unmapped area will define a **constant value mapping**. The following panel is invoked:

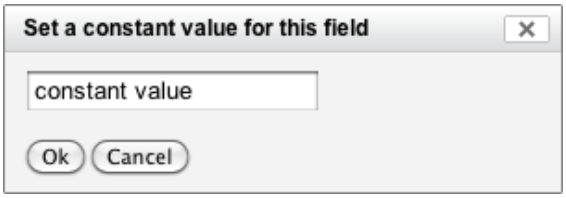

Figure 4.14 *ATHENA Metadata Ingestion Service - Constant Value Panel.*

The user can type a constant value in the provided text field. The value will then appear in the mapping area and in the result XML files. This type of mapping is useful for text that is intended to appear in all transformed items. Constant value mappings can be combined with XPath mappings to construct specific values such as URLs.

Conditions

Mappings can be restricted so that they will apply only under certain conditions. To define these conditions the button is used. This will allow the input of condition as shown in the following Figure:

Figure 4.15 *ATHENA Metadata Ingestion Service - Mapping Condition.*

The conditions supported are in the form of <Source XPath> = <Constant Value>. The corresponding mapping will apply only if the source XPath data for a specific item equals to the constant value provided. The condition source XPath can be set by dragging n' dropping a source element to the condition input area as shown in the previews Figure. The constant value can be set by double clicking on the constant value area.

If a more complex condition is required then the provided condition editor must be used by clicking on the icon next to the condition. An example of the condition editor is shown in the following figure:

Figure 4.16 *ATHENA Metadata Ingestion Service - Condition Editor.*

The condition editor provides the means to define more complex conditions. In addition to each clause's source element and constant value, the relational operator can be set. The logical operator that combines the clauses can also be defined. Additional clauses or sub-clauses can also be created as needed, by clicking the corresponding icon.

Preview & Mapping Summary

A summary of the defined mappings can be seen by clicking the 'Summary' button on the top of the mapping tool. This will invoke the following panel:

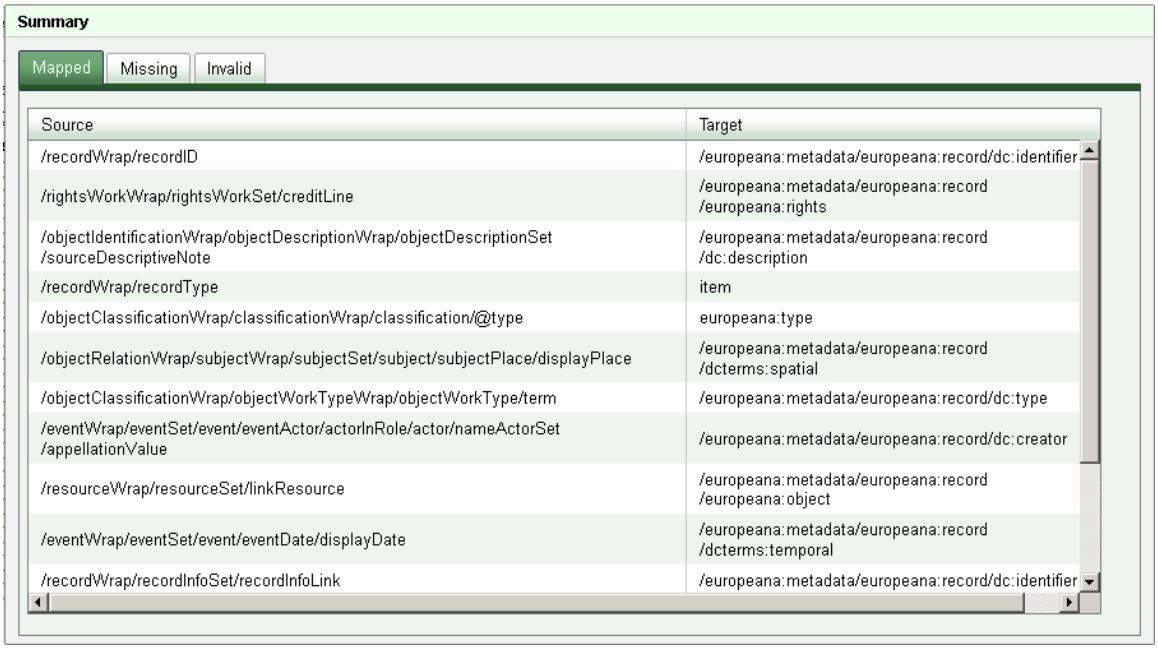

Figure 4.17 *ATHENA Metadata Ingestion Service - Mapping Summary Panel.*

This panel contains the following tabs:

- **Mapped:** All mapped source elements and the corresponding target elements.
- **Missing:** Mandatory target elements that have no mappings.
- **Invalid:** If a mapping definition was loaded based on another dataset, all XPaths form this dataset that do not exist in the current dataset appear in this tab.

The generated XSL can be previewed at any time by clicking the ‘Preview’ button on the top of the mapping tool. This will invoke the following panel:

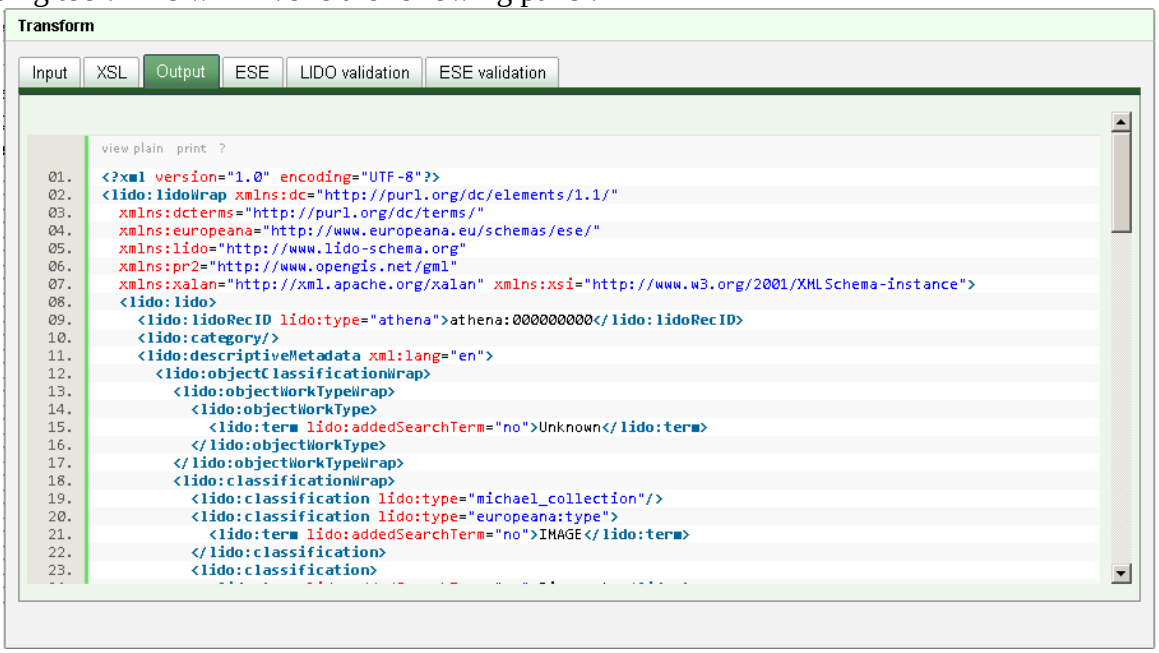

Figure 4.18 *ATHENA Metadata Ingestion Service - Preview Transform Panel.*

This panel contains the following tabs:

- **Input:** The XML that corresponds to the first item from the imported dataset.
- **XSL:** The generated XSL.
- **Output:** The item transformed to the LIDO schema based on the previous XSL
- **ESE:** The item transformed to ESE from LIDO
- **LIDO and ESE Validation:** Validation information for the transformed records. Any errors from wrong or incomplete mappings will be reported there.

Statistics

The statistics module facilitates the mapping and monitoring procedures. The following figure illustrates the statistics interface.

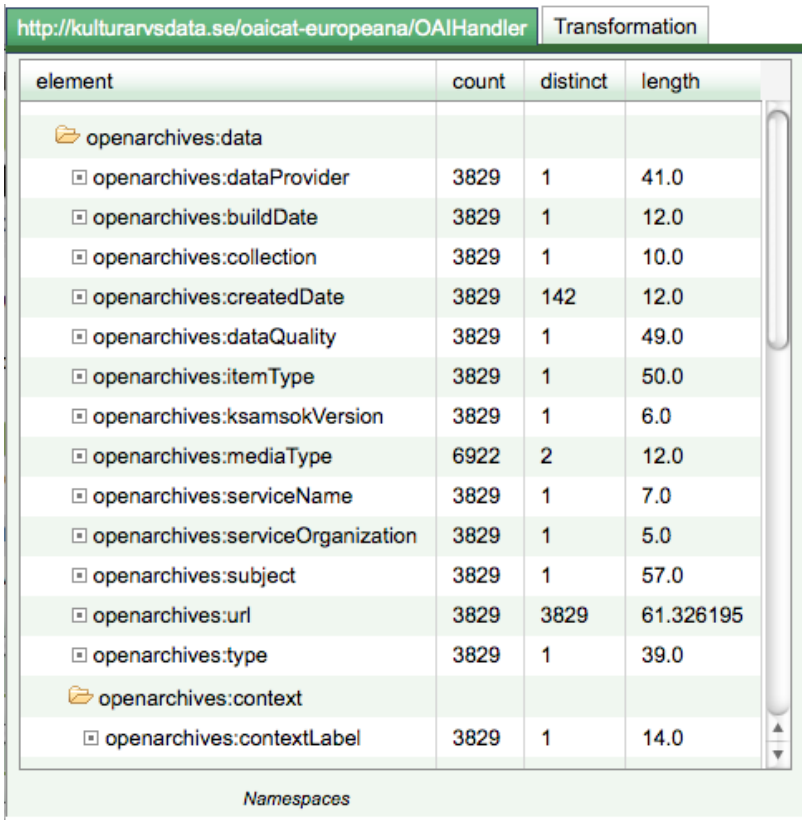

Figure 4.19 *ATHENA Metadata Ingestion Service - Input schema statistics.*

- Element. This is the name of the Element or the attribute of an element found in the import and that belongs to a specific XML Schema.
- Count. The number of times this element appear in the upload
- Distinct. The numbers of distinct/unique values the element or attribute holds.
- Length. The average length of the values the element or attribute holds.
  Figure presents the interface of the values of an element.
- Value. The rows under this column contain a distinct value found for a specific element or attribute.
- Frequency. The rows under this column contain the frequency of that specific value that appears in the preceding row.

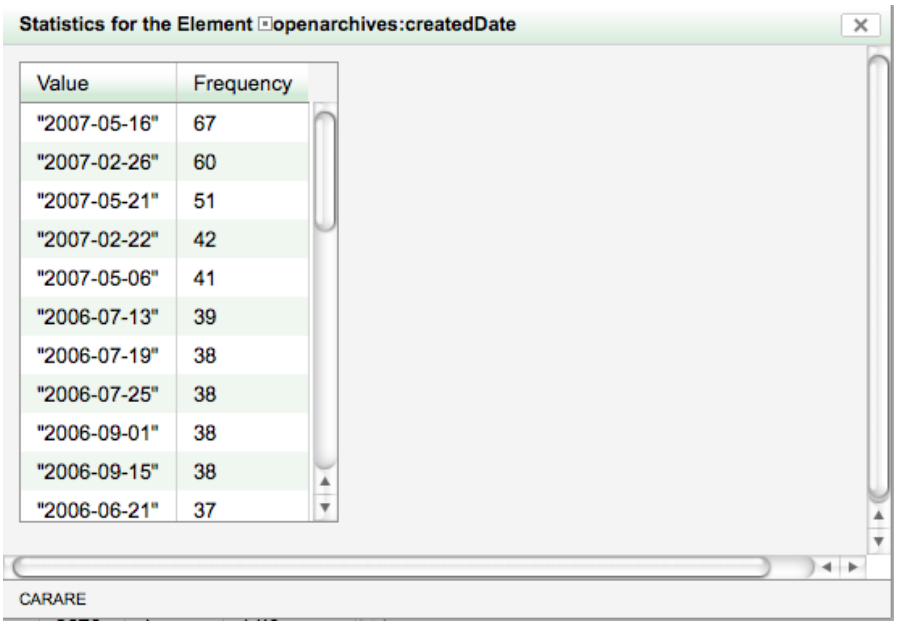

Figure 4.20 *Value distribution statistics for a specific element or attribute.*

4.3 Transformation Services

Transform

When the user has successfully defined the root and label elements for a specific "Import" and the mappings between the extracted source XML Schema and the target Schema of the system, he/she is able to perform the transformation of the data. For this to happen the user has to click the button which is visible under the extended view of a specific "Import" in the "Overview" tab. When this event is triggered by the user, he/she is presented with the modal window depicted in Figure . The user is presented with information regarding the different states a mapping might be based on the appropriate Icons. In order for the user to continue the transformation process he/she has to select a mapping from the drop down list and click on the "Submit" button.

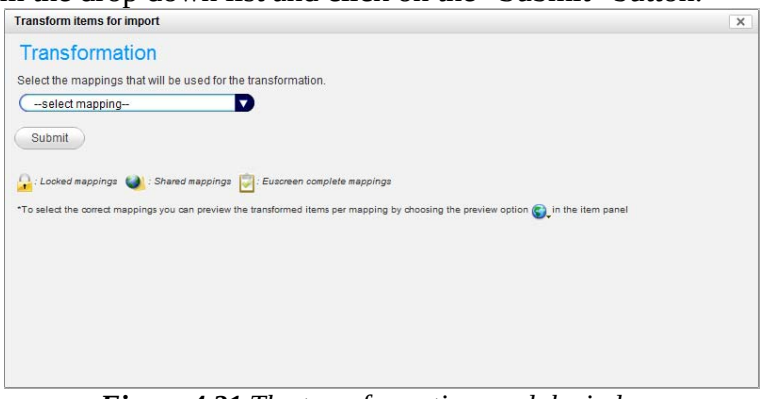

Figure 4.21 *The transformation modal window.*

In the case where the mapping is not correct, for example mandatory mappings are missing, the user is presented with a modal window explaining what the problems are, like the one depicted in Figure . This modal window has two distinct tabs presenting different kind of information to the user. The first tab presents any missing mappings to mandatory XPaths of the target Schema while the second one presents XPaths with erroneous mappings. The user is able to review the errors and then he/she has to go back and either select a different mapping or complete/correct the current select one.

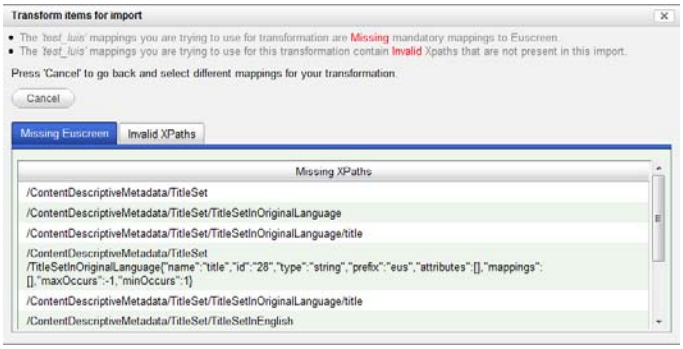

Figure 4.22 *The modal window presenting the errors found in a selected for transformation mapping.*

If everything goes well and the defined mappings do not contain any invalid XPaths or and there are no Missing mandatory mappings the user is redirected back to the "Overview" tab and an animated Icon takes the position of the button. When the user positions the mouse pointer over the animated icon, information regarding the progress of the transformation process is presented to him on a tooltip. Actually, while in the process of transformation, the system extracts each item XML instance based on the root element the user has defined and applies the XSLT transformation that is generated through the process of defining the mappings between the two Schemata, every generated item is then stored to the ingestion tool persistent data layer and is associated with the current Import. In the case where an error occurs in the process of transformation a "Red X" icon appears on top of the icon. When the user positions the mouse pointer on top of the icon he/she is able to review the errors that caused the transformation process to abort. In the case where the transformation ends without errors a "Blue" tick sign appears on top of the "transformation" icon and the user is able if he/she wishes to download the transformed items.

Review Transformed Dataset

After the completion of the transformation step in the ingestion tool core workflow, the user is able to review the original data together with the resulted transformed items and the generated XSLT on a per item level through the item browser in the "Review" tab. In order to do that the user has to press the for an individual item in the item browser. When this event occurs the user is prompted with a modal window where he/she is able to review the results of the whole process as depicted in Figure . The user is able to view the Input XML, the Invalid XPaths if any exist, the XSL generated in the mapping process, the output XML in the target Schema (in this case LIDO) of the system, the output XML and HTML rendering of the Publishing Schema (in this case ESE).

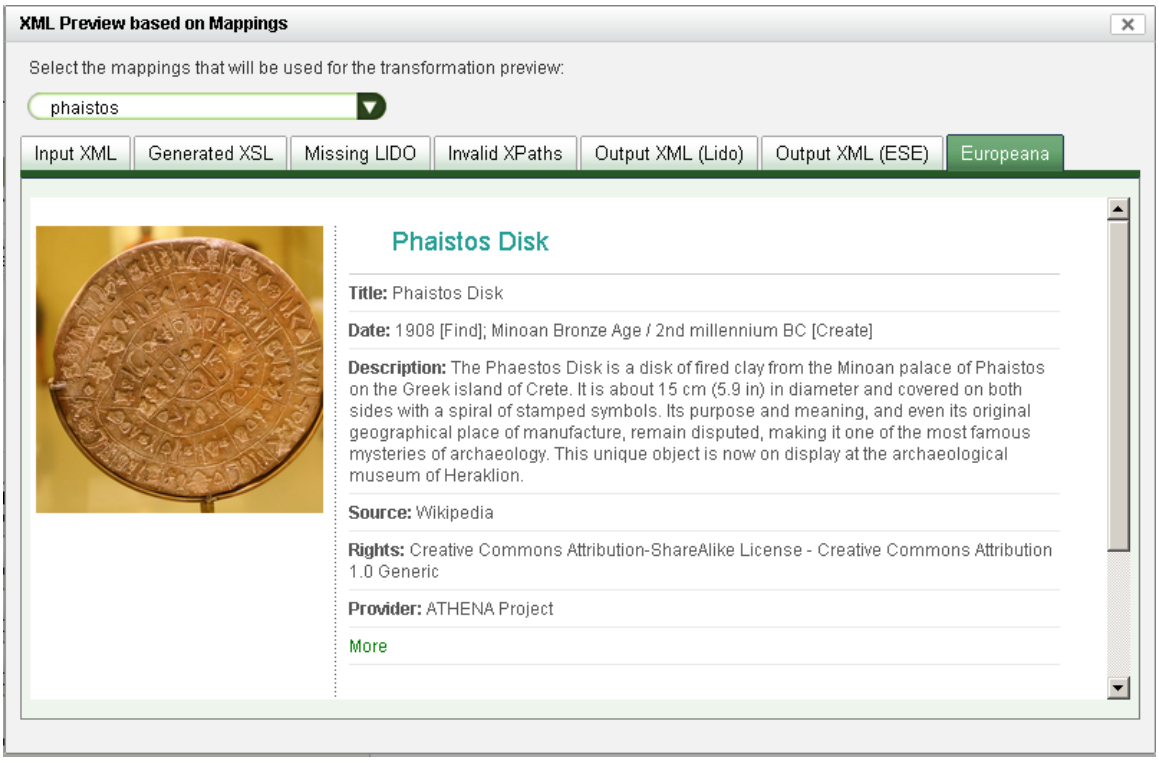

Figure 4.23 *The modal window where the user is able to review the results of a transformation on an individual item.*

Implementation Details

The system is implemented as a web service, where authentication is required to perform a series of tasks that correspond to work flow steps. The service is an application written in the Java programming language and hosted on a web server by the Tomcat servlet engine. Data is imported into a PostgreSQL database in xml format (as BLOB).

Once uploaded, the xml structure is parsed and represented in a relational database table. As this table can grow quite large it is partitioned into one partition per data upload. All data within one upload is treated as having the same structure, so it is not possible to upload different schemas (or more likely updated schemas) in one upload.

Most of the communication between the application and the database is implemented on the Hibernate framework, a high performance object/relational persistence and query service. This allows for powerful, yet simplified, management of housekeeping objects like Users, Organizations and Data Uploads while also providing additional functionalities such as integration with Lucene for indexing and querying data.

Once data is parsed into the relational table, indexes are built to allow quick access to any part or sub-tree of the xml-tree like data. These are currently constructed as PostgreSQL BTREE indexes; when content full text indexing is implemented, it will be based on Hibernate's search architecture. All further data manipulation such as mapping and transformation, normalization, enrichment, etc. is structured through the addition of extra tables annotating but not altering the original data. This allows easier comparison between uploads and facilitates the versioning strategy.

Functional Analysis

The mapping tool is designed using a client – server approach. A subset of the functionalities is implemented as server side services, while the user interface is rendered on the client inside a web browser. The communication between the client and the server is achieved using AJAX calls. One of the core design concepts of the mapping tool is that the user should be able to use all the functionality he might need in order to achieve the best possible result with minimal effort. In order to achieve that, the tool must be intuitive and visual aiding and appealing. Another important design concept is that performance must be ensured because the Ingestion platform must be able to perform computational intense tasks, e.g. metadata transformation and data parsing, without affecting the interaction between the user and the web service. This is achieved in a great degree by separating the interface rendering and the interaction with the user from intense tasks that are executed on the server side. At the same time the communication overhead between the client and the server was minimized as much as possible.

The XML Schema Parser sub-module is responsible for parsing the target XML Schema and retrieves any valuable information it's stored in its structure, e.g. annotations used for documenting the schema. After parsing the XML schema the sub-module generates an intermediate data structure serialized using the JSON language in order for the user interface to parse and generate the corresponding visual components. The rationale behind choosing JSON as the serialization language for that data structure is the software interoperability the Ingestion platform is attempting to achieve. JSON is a well supported language with interpreters for every major language platform available which reduces the overhead introduced by using XML for exchanging messages both by a reduced memory footprint needed and a simpler structure which makes parsing a much easier and lightweight task. The XML Schema Parser sub-module requests and retrieves the schema needed from the persistent data layer. By using Hibernate for accessing and manipulating the data model, the software's architecture ensures the platform neutrality and separates the maintenance of the sub-modules from that of the data model itself.

The XML Schema parser sub-module is based on the XML Schema Object Model (XSOM) API that is part of the JAXB API for XML data binding. The main design goals of the XSOM API are a) to expose all the defined in the schema spec and b) to provide additional methods that help simplifying client applications. XSOM consists of roughly three parts; the first part is the public interface the entire functionality of XSOM is exposed by this interface to the client. The second part is the actual implementation of these interfaces. Finally the third part is a parser that reads XML representation of XML Schema and builds the XSOM data model accordingly. This part of the code is mainly generated by the RelaxNGCC API.

For the needs of the ATHENA Metadata Ingestion service, an import is not required to include the schema used. This simplifies the actual work for the user and at the same time the set of schema components that have to be mapped is reduced to only those that are used, thus reducing redundancy. The Schema Generator sub-module produces the required simplified version of the schema that corresponds to a specific import by the user. When a user triggers the invocation of the mapping tool for a specific import, this sub-module is also invoked. It communicates with the data layer using the Hibernate persistent API. The next step in the workflow of the Schema Generator sub-module is to parse the data for a specific import and generate a tree like structure using HTML elements that represents the schema used. This tree like structure is then transmitted to the User Interface sub-module and is enhanced using JavaScript in order to create an interactive tree that represents a snapshot of the XML schema that the user is going to use as input for the mapping process.

The User Interface sub-module is responsible for creating and presenting an intuitive and visual appealing environment for the user to define mappings, without sacrificing any of the functionality needed to properly achieve the task of schema mapping. This sub-module is invoked by the user through the Overview interface of the Ingestion platform per import listed there. When the invocation occurs the server retrieves the id of the import and the workflow of the mapping tool is executed; the final step of that workflow is the transmission of all the appropriate structures to the user's browser where the mapping tool is rendered. The User Interface sub-module is implemented in JavaScript using the YUI library from Yahoo. The usage of that library for implementing the visual components also ensures cross-browser compatibility.

5. ATHENA OAI-PMH Server for Europeana Harvesting

As it was designed in the ATHENA ingestion workflow and outlined in DE7.1, the following hierarchy exists regarding metadata ingestion, harvesting and mapping:

1. The ATHENA Metadata Ingestion Server is used to make the ingested metadata accessible for mapping to the ATHENA reference model, LIDO.
2. Then the ingested metadata records are transformed to LIDO records and are aggregated in the ATHENA repository.
3. The ATHENA repository uses an OAI_PMH Server to transform and publish the ESE versions of the metadata records for the Europeana Ingestion Office.

5.1 Introducing OAI-PMH

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is a low barrier mechanism for repository interoperability. Data providers are repositories that expose structured metadata via OAI-PMH. Service providers then make OAI-PMH service requests to harvest that metadata. OAI-PMH is a set of six verbs or services that are invoked within HTTP[8]. In the context of the ATHENA project, OAI-PMH provides a mechanism for interoperability between the Ingestion Tool and various other modules or platforms. E.g. using the OAI-PMH terminology, the Ingestion Tool constitutes the data provider while Europeana, or any third party portal, are service providers that are able to request and retrieve metadata records via the OAI-PMH verbs or services.
The ATHENA Metadata Ingestion Tool is capable for managing heterogeneous collections of metadata records while exposing services for mapping and transforming from one metadata schema to another. In order to extend the functionalities of the ATHENA Metadata Ingestion Tool with the OAI-PMH protocol and thus to expose Metadata through an interoperable mechanism, the defined OAI-PMH verbs are implemented on top of the underlying and domain specific data layer. An issue that arises is that while being able to manage collections of metadata records, the OAI-PMH verbs operate on an item level, something that complicates the implementation of the appropriate verbs directly on top of the Ingestion Tool data layer. For this reason and to also include a set of additional functionalities that are not directly associated with both the Ingestion Tool and the OAI-PMH protocol, it was decided to follow a technical approach in which an exporting mechanism exist between the Ingestion Tool platform and another data repository more suitable for the needs of an OAI-PMH data repository.

[8] http://www.openarchives.org/pmh/

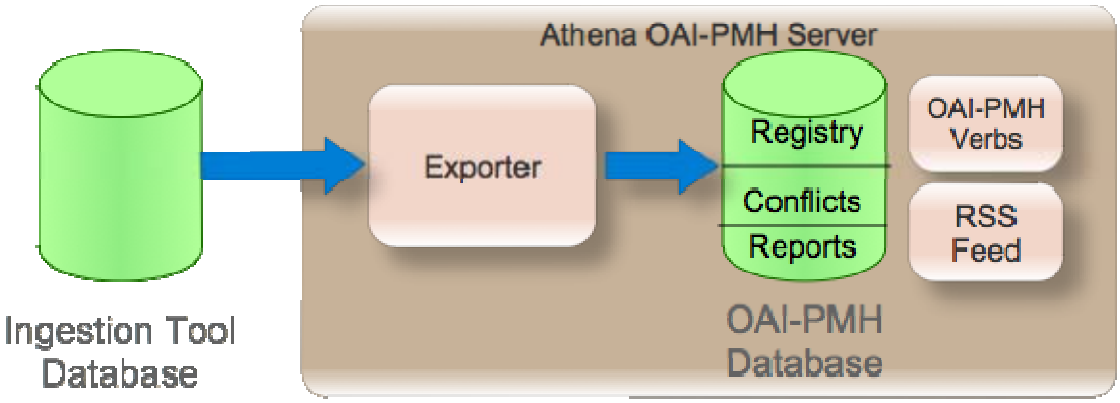

Figure 5.1 *The ATHENA Metadata Ingestion OAI-PMH Server*

Based on the architecture presented in Figure the ATHENA OAI-PMH Platform consists of an appropriate OAI-PMH database together with a set of services, these are the following:

- Exporter: a mechanism that is responsible for iterating the imports that are stored in the Ingestion Tool database, transform them in the appropriate format of the OAI-PMH database and store them.
- OAI-PMH database: constitutes the data layer of the ATHENA Metadata Ingestion OAI_PMH Server where the metadata records that will be exposed via the OAI-PMH verbs are stored.
- OAI-PMH Verbs: the various verbs that are defined by the OAI-PMH v2.0 protocol for exposing metadata records through HTTP are implement in such a way that expose directly and without transformation the metadata records that are stored in the OAI-PMH database.
- RSS Feed: an implementation of the Atom protocol for exposing automatically information to subscribers regarding the activity of the ATHENA Metadata Ingestion OAI_PMH Server. E.g. every time a new data set is imported a notification is sent through the RSS Feed to all the subscribers.

In the following chapter, a more detailed technical description of the various modules of the ATHENA Metadata Ingestion OAI_PMH Server implementation will be presented.

5.2 Analysis of the ATHENA Metadata Ingestion OAI_PMH Server Architecture and Implementation

A core characteristic of the ATHENA Metadata Ingestion Tool is that of being agnostic in regard to the Schema of an imported dataset, a characteristic that is also inherited to the ATHENA Metadata Ingestion OAI_PMH Server. For this to be achieved the data layer of the platform has to be able to handle heterogeneous metadata Schemata. Based on this assumption, it was decided to implement the underlying data layer using a NoSQL solution that does not enforce any particular schema and thus it will be able to adapt to metadata records that conform to different Schemata. The NoSQL solution that was used for the ATHENA OAI_PMH Server is the MongoDB[9] document database. MongoDB is designed around the concept of documents that are internally implemented as JSON[10] document and internally stored using the BSON[11] format which stands for a binary

[9] http://www.mongodb.org/

[10] http://www.json.org/

serialization of the JSON format. It allows the existence of JSON documents in the same database or even collection having different fields and thus it does not enforce any specific data model schema. Finally it provides a rich set of native implementations of drivers[12] for communicating with the database while the JSON format provides added value in the development of web application because the stored data do not have to be transformed to a different format in order to be consumed by the applications.

The ATHENA Metadata Ingestion OAI_PMH Server Data Layer is designed around three distinct collections that exist inside a MongoDB data base, collections can be perceived as tables of typical SQL databases, although it is not required that each document conforms to a specific datamodel and set of fields. These collections are the following:

1. Registry: in this collection the actual metadata records are stored and accessed by the implemented OAI-PMH verbs.
2. Conflicts: every time a new dataset is imported in the OAI-PMH repository, it is checked for the existence of duplicate records, if any exist, they are reported and stored in a different collection while they are also associated with a specific Report document.
3. Reports: every time an operation occurs on the OAI-PMH Repository, a report is created which includes any useful information regarding the operation. For example, in the case of import if any conflict is identified between the items it is reported for further reference.
4. The Registry collection constitutes the core collection of documents for the OAI-PMH repository. It contains all the records that it is desired to be exposed via the OAI-PMH repositories. Each record is stored inside a JSON document which also contains additional information useful to the platform and the aggregation repository. More specifically, the record document contains information regarding the organization to which the item belongs to, a unique hash key that is generated by calculating the SHA1 hash of the string representation of the item, a datestamp which represents the date and time the record was inserted and finally a namespace “prefix” value which is required by the OAI-PMH specification. An example of a Registry Document instance is depicted in Figure .

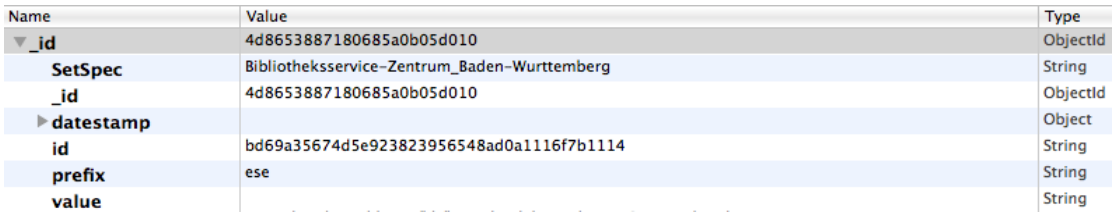

Figure 5.2 *Structure of the Registry Document.*

As it was mentioned earlier, another important collection of documents that is stored as part of the OAI-PMH repository is the Reports collection. The documents that are stored in this collection represent a set of valuable information that corresponds to specific actions of the repository platform. These actions are stored as values in the type attribute of the document and take one of the following values:

1. Add: this type represent an addition action in which records are added in the registry.
2. Update: this type represents an update action in which a set for a specific import already exist and it is updated by adding new metadata records.
3. Delete: this type represents the action of deleting a number of records from a specific import that already exists in the registry.

[11] http://bsonspec.org/#/specification

[12] http://www.mongodb.org/display/DOCS/Drivers

Apart from the action type a number of other values are also stored as part of the Report Document. More specifically, a set of valuable statics are stored; the number of conflicted items that were identified, the total number of the inserted records and the total number of items which corresponds to sum of the inserted records plus the conflicts. Two datestamps are also stored as part of the Report document, one datestamp corresponds to the time of creation of the document and the other to the time of closing of the document, in this way it is possible for the repository to calculate the time it took to import a whole data set into the database. Finally the date the import was published to the Ingestion Tool is stored together with the name of the organization it belongs to. A visual representation of the Report document structure is depicted in Figure .

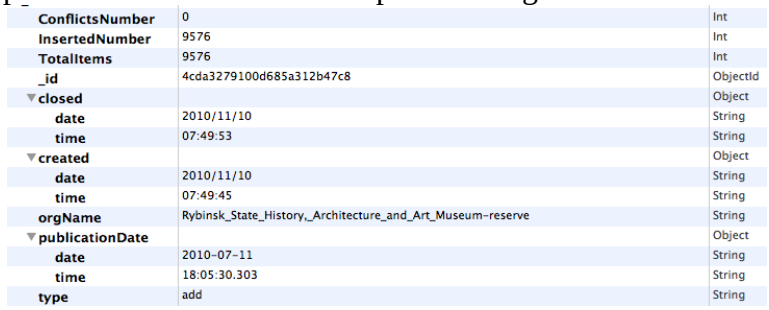

Figure 5.3 *Structure of the Report document.*

The last collection of the OAI-PMH repository database is the conflicts collection. The documents stored in this contain any metadata records that at the time of the exporting of an import from the Ingestion Tool to the OAI-PMH repository were found to be conflicted. These documents are quite simple in their structure. They contain an SHA1 hash of the conflicted item that was found together with the record and a reference to the Report document that it belongs to. In this way it is possible for someone to browse the actions that were made on the repository (e.g. additions, deletions and updates) and directly view the items that were found as conflicted for the cases of additions and updates. An example of a conflict document is depicted in Figure 5.1. It has to be noted that the whole procedure of creating unique hash codes and identifying conflicted items is an important functionality of the ATHENA OAI-PMH repository platform because it provides a mechanism for creating unique ids for the metadata records and also a mechanism for identifying duplicate.

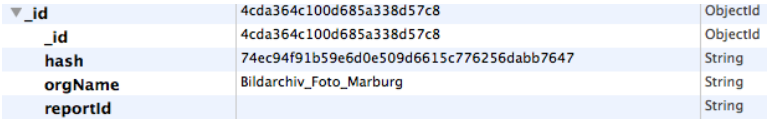

Figure 5.1 *Structure of the Conflict document*

On top of the data layer that was described so far a set of functionalities is built and more specifically an RSS Feed based on Atom and of course the implementation of the actual OAI-PMH verbs. The implementation of the verbs is based on the customization of the oaicat[13] API which provides an abstract implementation of the OAI-PMH v2.0 specified verbs that can be customized in order to operate on top of different data layer technologies (e.g. flat XML files, relational databases etc.). The verbs that were implemented in order to work with the current OAI-PMH Repository implementation are the following:

- Identify: this verb provides basic information regarding the running instance of the OAI-PMH v2.0 data repository such as, contact details of the admin of the repository, the base url that can be used by a harvester, a sample of an identifier among others. For a complete list of the information provided by the Identify verb someone can refer to the OAI-PMH

[13] http://www.oclc.org/research/activities/oaicat/default.htm

specification. The information served by this verb does not have to be stored in the underlying data layer but is part of the configuration files of the verbs implementation.

- GetRecord: given the identifier of a record and the desired namespace prefix, this verb fetches from the MongoDB database the corresponsive Metadata record and delivers it as a response to the harvesting client. In the current implementation a query is executed based on the prefix which is part of the Registry Document and the unique id which is generated by the SHA1 hashing of the initial Metadata Record, this query corresponds to an exact match on the database.
- ListIdentifiers: given a set name and a namespace prefix, this verb responds with a list of identifiers of items that correspond to these criteria. The set name is identical to the name of the organization. In this way it is possible to organize the records of the repository around organizations/providers. Again, the namespace prefix is matched with the prefix field of the Registry Document.
- ListRecords: this verb operates in a similar way to the ListIdentifiers with the main difference being that instead of returning only the identifiers, the complete Metadata Record is served.
- ListMetadataFormats: this verb is implemented by aggregating all the unique prefix values that are stored in the data layer by executing an aggregation for uniqueness query on the Registry collection. The resulting response that is served by this verb contains all the unique namespace prefixes that exist in the OAI-PMH repository and can be used for accessing the Metadata Records.
- ListSets: this verb returns a list of all the sets that exist in the OAI-PMH repository. Sets are named after the organizations that provide metadata records to the OAI-PMH repository, in this way it is possible for someone to retrieve only the records that are associated with a specific organization. The way the values are extracted is similar to the ListMetadataFormats verb.

In every case that is needed, the verbs are implemented in such a way that they support paging through the mechanism of resumption tokens as it is defined by the OAI-PMH specification. The number of the returned items is specified through the configuration files of the oaicat running instance. Finally, by being a servlet implementation, the oaicat specific instance can be served through any of the available servlet contains that exist, e.g. tomcat, jboss, jetty etc. Currently it is served via a running Apache tomcat instance.

The RSS feed is implemented following the Atom Syndication Format[14] which is an XML language used for web feeds while the Atom Publishing Protocol (APP) is a simple HTTP based protocol for creating and updating web resources. The purpose of an RSS feed in the current OAI-PMH repository implementation is to provide a mechanism for notifying metadata consumers for the occurrence of specific actions, for example when new items are added or updated to the repository in an automatic way. This is achieved by creating the RSS Feed on top of the Reports collections that was described earlier, in this way every time a new report is generated the feed is automatically updated and the subscribers are informed for the associated action. The implementation of the RSS Feed service is based on the Apache Abdera project[15]. The goal of the Apache Abdera project is to build a functionally-complete, high performance implementation of the IETF Atom Syndication Format (RFC 4287) and Atom Publishing Protocol (RFC 5023) specifications. The current implementation of the RSS Feed which is based on the Apache Abdera project, is built by

[14] http://tools.ietf.org/html/rfc4287

[15] http://projects.apache.org/projects/abdera.html

communicating directly with the MongoDB based data layer of the OAI-PMH repository, everytime a new Report document is inserted, it is also transformed into the Atom protocol XML representation and published on the Atom Feed that is maintained through the API of Apache Abdera. Finally, an Apache Abdera instance can be served via any of the available servlet containers, but in the current implementation it is contained in a Jetty servlet container for performance reasons.

The last module of the OAI-PMH Repository is the exporting API which is responsible for fetching data from the Ingestion Tool, performing any needed processing on the metadata records and storing them into the OAI-PMH Repository data layer. As it was also mentioned in the previous chapter, the requirements of these two platforms are different in the sense that the OAI-PMH Repository operates on a record level while the Ingestion Tool is designed and implemented to operate on a record collection level. For this reason it is necessary to equip the repository with a mechanism capable of dealing with the differences between the two requirements sets. The exporting API is capable of iterating the list of published data sets on the Ingestion Tool and imports them into the OAI-PMH Repository. The workflow that is executed is the following. Initially, the Exporting API iterates all the records that exist on the publication table of the Ingestion Tool. For each publication it checks whether an import to the OAI-PMH Repository exist and if yes then it checks if it needs an update. In any case after it is decided that items from the publication will be imported in the OAI-PMH Repository, it start iterates the records of the publication, processes them and stored them in the MongoDB database and more specifically in the Registry collection. Also, at the initial step of execution of the Exporting API, a new Report Document is created and its fields are updated as long as the processing of the current publication is executed. For each record that is parsed from the publication an SHA1 hash is generated and by querying that data layer of the repository, it is possible to identify if it conflicts with another record that is already stored. In this case, a conflict document is generated and stored in the conflict Collection. By doing that, it is guaranteed that at the end the OAI-PMH Repository will contain only unique items that will served through the OAI-PMH verbs that were described earlier. Finally, the execution of the Exporting API can be scheduled by using the Quartz scheduler API[16].

[16] http://www.quartz-scheduler.org/

6. Conclusions

Present report documents the establishment of interoperability between the ATHENA aggregation system and repository and the Europeana Harvesting Office. The semantic interoperability platform that was designed and implemented within WP7 of the ATHENA project is outlined in accordance with the ingestion implementation plan and Europeana's modelling efforts. The web service is used to ingest metadata from a diverse group of cultural heritage content providers, to homogenise and align them with an established metadata schema standard that guarantees semantic interoperability and, to publish them in the Europeana Semantic Elements (ESE) schema for harvesting and presentation in the Europeana portal. It is based on a platform developed by the leader of the WP NTUA, which is customised and updated according to ATHENA requirements and the most recent developments in the Europeana family of projects. The tool was presented and tested by a wide variety of ATHENA users that attended respective WP7 workshops and is in an ongoing process of incorporating additional functionalities based on user feedback and recent developments in the field of digital cultural heritage and the web of data. Ingestion was continuously supported for the duration of the project and will be extended for at least a year after its end. The tool was also used for prototyping of the new Europeana Data Model and respective ingestion services[17].

[17] http://europeanalabs.eu/wiki/EDMPrototyping