

LUISA

*Learning Content Management System Using Innovative Semantic Web
Services Architecture*

IST- FP6 – 027149



Deliverable 3.3.2 Annotation Tool

Fredrik Enoksson
Matthias Palmér
Mikael Nilsson
Ambjörn Naeve

Due date of deliverable: 29/02/2008

Actual submission date: 13/03/2008

Start date of the project: 01/03/2006

Duration: 30 Months

Fredrik Enoksson
University of Uppsala

Version 2, dated 13/03/2008

Change History

Version	Date	Status	Author (Partner)	Description
0.1	20070117	Initial draft	Fredrik Enoksson (ULL)	Table of contents and the first version of the chapters
0.2	20070131	Initial draft	Fredrik Enoksson (ULL), Matthias Palmér (ULL)	Finalizing the chapters and the whole document for the internal review.
0.3	20070301	Initial draft	Fredrik Enoksson (ULL), Matthias Palmér (ULL), Ambjörn Naeve (ULL)	Extending all chapters in order to improve readability and also taking into account the internal review.
1.0	20070302	draft	Fredrik Enoksson (ULL), Matthias Palmér (ULL)	Finalizing the draft.
1.1	20070905	draft	Matthias Palmér (ULL), Ambjörn Naeve (ULL)	Incorporating changes suggested by the commission review.
1.2	20080220	draft	Fredrik Enoksson (ULL), Mikael Nilsson (ULL), Matthias Palmér (ULL), Ambjörn Naeve (ULL)	Changes done to fit the description of the Annotation Tool as it looks now
1.3	20080228	draft	Fredrik Enoksson (ULL)	Version sent for internal review
2.0	20080309	final	Fredrik Enoksson (ULL), Matthias Palmér (ULL)	Adjustment made according to the internal review



EXECUTIVE SUMMARY

This deliverable describes the LUISA Annotation Tool and the Annotation Tool library, which is the code library that the tool is built upon. The Annotation Tool library has been designed in order for a programmer to easily build a specific Annotation Tool and integrate it into a specific setting, where the resulting user interface lets the user annotate metadata in a form based manner. The form is built from a form model that is used to tailor specific user interfaces. The form model is in turn created from a configuration mechanism called Annotation Profiles that were described in Deliverable 3.2 – *Annotation Profile Specification*. With this configuration mechanism new forms, for editing new kinds of metadata, are created from a (new) Annotation Profile. This means that no further development of the code of the Annotation Tool is necessary when new metadata elements are needed. The scope of this report is on the tool itself and the code library, rather than how end user should use it, therefore the main focus is on three different kinds of developer roles, *Code Library Developer*, *User Interface developer* and *User Interface Integrator*. For the *Code Library Developer* the design choices are described and the main ideas of how the Annotation Profile are processed in order to create a form. This part also includes information for the *User Interface Developer* which has the task to tailor new user interfaces / skins for the annotation tool. The role of the *User Interface Integrator* is to integrate a user interface /skin of the Annotation Tool into a bigger application in need of annotation support. The integration into the rest of the LUISA architecture is described in the last sections of the report. The specific LUISA Annotation Tool required a distributed solution where the tool updates a metadata-repository on a remote basis.

Document Information

IST Number	Project FP6 – 027149	Acronym	LUISA
Full title	Learning Content Management System Using Innovative Semantic Web Services Architecture		
Project URL	http://www.luisa-project.eu		
Document URL			
EU Project officer			

Deliverable	Number D3.3.2	Title	Annotation Tool
Work package	Number 3	Title	Learning Object Annotation

Date of delivery	Contractual	29/02/2008	Actual	13/03/2008
Status	Version 2, dated 13/03/2008		final	
Nature	Report <input checked="" type="checkbox"/>	Demonstrator <input type="checkbox"/>	Other <input type="checkbox"/>	
Dissemination Level	Public <input checked="" type="checkbox"/>	Consortium <input type="checkbox"/>		
Abstract (for dissemination)	The main ideas and the structure of the Annotation Tool developed in WP3 is described in this document. The annotation tool is designed to be adaptable and easily integrated into different settings and the document is aimed at describing the Annotation Tool for three different kinds of developer roles: developers of the core library, developers of user interface and user interface integrators.			
Keywords	Annotation Tool, RDF, Semantic Web, Graphical User Interface, Remote Editing, Metadata			

Authors (Partner)	Fredrik Enoksson (ULL), Matthias Palmér (ULL), Mikael Nilsson (ULL), Ambjörn Naeve (ULL)		
Responsible Author	Fredrik Enoksson	Email	fen@nada.kth.se
	Partner ULL	Phone	+46 18 471 62 90

Project Consortium Information

Partner	Acronym	Contact
Atos Origin S.A.E. (Coordinator)	ATOS 	Nuria de Lama Atos Origin S.A.E. c/ Albasanz 12 E-28037 Madrid, Spain Email: nuria.delama@atosorigin.com Tel.: +34 91 214 9321 Fax: +34 91 754 3252
University of Alcalá de Henares	UAH 	Dr. Miguel-Angel Sicilia Information Research Unit University of Alcalá Ctra. De Barcelona, Km 33.6 E-28871Alcalá de Henares (Madrid), Spain Email: msicilia@uah.es Tel.: +34 91 886 6603 Fax: +34 91 885 6646
University of Uppsala	ULL 	Dr. Ambjorn Naeve University of Uppsala Kyrkogårdsgatan 2 C Uppsala Email: amb@nada.kth.se Fax: +46 184-716-294
Open University	OU 	Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom Email: j.b.domingue@open.ac.uk Tel.: +44 1908 655014 Fax: +44 1908-653-169
University Henri Poincaré	UHP 	Dr. Monique Grandbastien University Henri Poincaré Vandoeuvre les Nancy 54506, PO Box 239, France. Email: monique.grandbastien@loria.fr Fax: : +33 383-278-319
Giunti Labs S.r.l.	GIUNTI 	Fabrizio Giorgini Giunti Labs S.r.l. Abbazia dell'Annunziata Via Portobello Baia del Silenzio 16039 Sestri Levante (GE), Italy Tel.: +39.0185.42123 Fax: +39.0185.43347
EADS FRANCE – Innovation works	EADS 	Anne Monceaux EADS FRANCE – Innovation works Avenue Didier Daurat - Centreda 1, Toulouse, 31700, France. Email: anne.monceaux@eads.net Tel.: +33 561-168-825 Fax: +33 561-187-611



TABLE OF CONTENTS

EXECUTIVE SUMMARY	3
1 INTRODUCTION	7
1.1 Abbreviations Used in this Document	9
2 DESIGN CHOICES OF THE ANNOTATION TOOL LIBRARY	10
2.1 Design Overview	11
2.2 Creating a form	12
2.2.1 Creating the Variable Bindings	14
2.2.2 Creating and using the Form Model	16
2.2.3 Guidelines for Presenting Multiplicity	17
3 DESIGN AND ARCHITECTURE OF THE LUISA ANNOTATION TOOL	18
3.1 A typical editing scenario	18
3.2 Techniques used for the GUI	19
3.3 The interface to the end-user	20
3.4 Communication between the client and the server	21
3.4.1 Retrieve a list of resources	22
3.4.2 Create a new metadata record for a resource	23
3.4.3 Retrieve a form template	23
3.5 The server side and communication to the remote repository	25
3.5.1 Constructing a query to retrieve appropriate subgraph	26
3.5.2 Handling the retrieved subgraph	27
3.5.3 The remote repository	27
4 ENCOUNTERED PROBLEMS	29
4.1 Datatypes	29
4.2 Additional information for translating to WSML	30
5 CONCLUSION	31
REFERENCES	32

1 INTRODUCTION

This document describes the Annotation Tool (AT), developed for LUISA in work package 3, from a technical point of view. The LUISA Annotation Tool is a specific user interface built on top of the Annotation Tool Library¹, a code library that is constructed to work with Annotation Profiles that have been specified in LUISA deliverable 3.2 – *Annotation Profile Specification* [1]. An Annotation Profile is a configuration mechanism from where a form-based editor of RDF metadata can be automatically generated. The profile defines what metadata constructs that are possible to edit and in what way, ie outlining the form by telling if text-fields or a predefined set of choices should be used.

An important goal of the development of the Annotation Tool Library is to make it easily reusable in other applications that need to construct their own specific user interface for metadata editing. Applications, including the LUISA Annotation Tool, that are based on the Annotation Tool Library (and indirectly upon Annotation Profiles), can easily change the set of metadata elements used, without having to redesign or further develop the application. Hence, the use of Annotation Profiles simplifies the development of more advanced editing scenarios as it provides editors as ready made building blocks.

Furthermore, the Annotation Tool library is carefully designed to be easily integrated as a module into a surrounding application. This requires a technical adaption as well as a change in the appearance of the Annotation Tool to get a unified look and feel. In the rest of this document the term **Annotation Tool** will refer to any implementation using the Annotation Tool library and the term **LUISA Annotation Tool** will refer to the specific implementation done for LUISA.

In LUISA, both the industrial and the academic use-case need to allow different users to edit different sets of metadata elements depending on their role in the editing process of a resource. The academic use-case requires librarian, teacher and administrator roles while the industrial use-case requires LO author, administrator, teacher, learner, training manager and team manager roles. Hence, providing specific support for all of these roles within the corresponding domain would lead to a total of 9 different annotation tools to be developed and integrated within corresponding authoring environment of each use-case. With the design of the LUISA Annotation Tool, it will be sufficient to specify 9 different annotation profiles and do the integration into each use-case authoring environment only once.

For a longer discussion of existing tools and the benefits of configurable annotation tools that are reusable, see deliverable 3.1 – *State of the art* [2].

1 A further development of the SHAME library (<http://kmr.nada.kth.se/shame>) created by the KMR-group at ULL

The Annotation Tool fits into the proposed LUISA architecture according to figure 1 below

The Annotation Interface in figure 1 is the user interface of a specific annotation tool and can be built into other application like a Learning Management System (LMS) where the tool communicates with the LOMR (Learning Object Metadata Repository) for retrieval and storage of data.

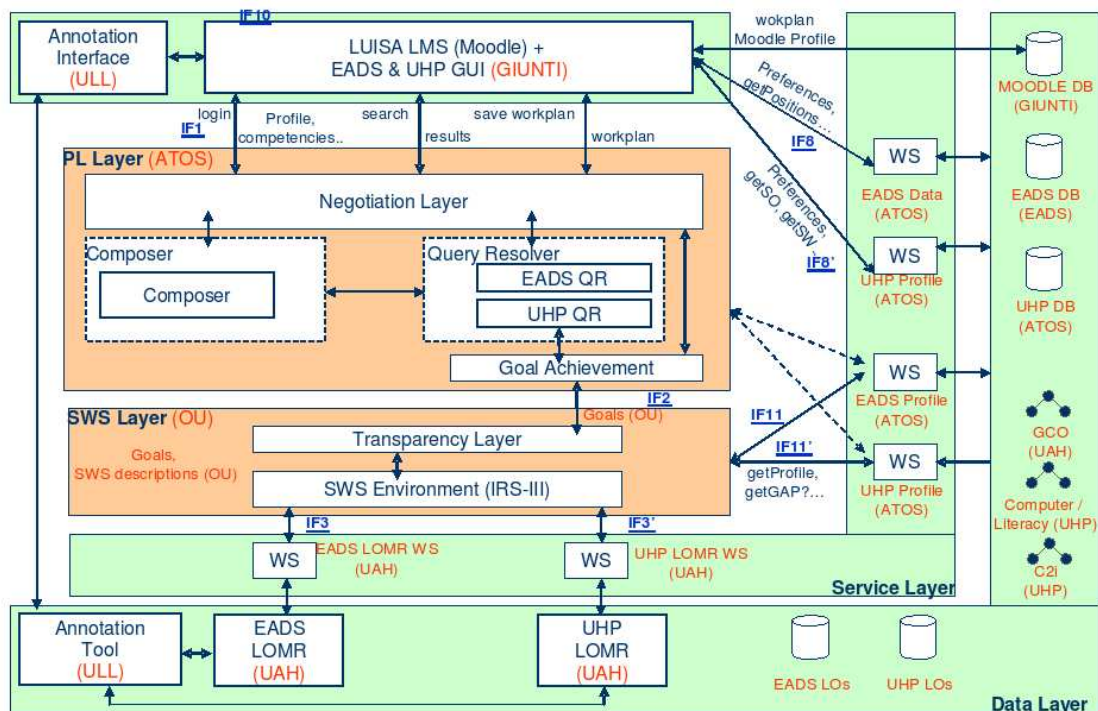


Figure 1: The proposed LUISA architecture, where the annotation tool can be seen in the left part

In deliverable 3.2, three roles when using Annotation Profiles for editing was introduced: The *Annotation profile author*, the *Annotation profile facilitator* and the *end user*. Focusing on the development of the Annotation Tool library and applications built upon that library, we see three additional developer roles.

1. **AT Library Developer** – Develops the Annotation Tool library used by the annotation tool
2. **AT User Interface Developer** – This role includes developing interfaces to the AT with the help of the AT library
3. **AT User Interface Integrator** – In this role an interface developed by the interface developer is put into another setting, for example a standalone application or a web-setting

This document is aimed at providing useful information for these three roles, where section 2 describes the design choices made for the code library (role 1)



and also how the form model is constructed and can be used (role 2). Section 3 outlines a typical integration (role 3) into a web-setting which is the specific LUISA Annotation Tool. Furthermore, when implementing the specific LUISA Annotation Tool some problems occurred that will be covered in section 4.

1.1 Abbreviations Used in this Document

AJAX – Asynchronous JavaScript And XML

AP – Annotation Profile

API – Application Programming Interface

AT – Annotation Tool

GUI – Graphical User Interface

RDF – Resource Description Framework

REST – Representational State Transfer

SPARQL – SPARQL Protocol and RDF Query Language

WSML – Web Service Modeling Language

2 DESIGN CHOICES OF THE ANNOTATION TOOL LIBRARY

The Annotation Tool library is designed to make it possible to create editors where a user can edit metadata in a form-based manner. The metadata the user is editing with a specific Annotation Tool will be stored in RDF, but this underlying encoding is kept transparent to the user. A specific form is created from an Annotation Profiles (AP), that are created in advance, and act as input to a specific AT. The ontologies in LUISA are created in WSMML, but to edit the metadata with a profile based approach a query language is needed. No established query languages exist for WSMML, so instead of creating a query language or using an unestablished one, the RDF representation of the WSMML ontologies will be used instead. To query WSMML by using the RDF representation was also recommended in [3]. The overall design goal of the Annotation Tool library is to be as generic as possible, that is why the following choices were made:

- **Flexible use of AP syntax** – An AP could be expressed in more than one way, therefore the AT was designed to make it easy for developers to add support for new syntaxes. The current version of the AT library has been implemented to support several specific syntaxes, and adding yet another syntax should be done fairly easy.
- **Work with any RDF-API** – The easiest way to handle RDF would be to use an established / standardized generic API for Java. Unfortunately, no such API exists at the moment, but two fairly stable Open Source implementations exist: *Jena*² (that has been developed by HP) and *Sesame*³. The AT library uses Jena, but has been designed with the intention to make an easy switch to any RDF-API.
- **Express syntax for RDF in several ways** – RDF can be serialized in several ways, where the most popular are N3⁴ and RDF/XML⁵. With the AT library it is possible to output RDF in both those syntaxes, and the idea is to support any kind of serialization of RDF.
- **Well defined Form Model API** – Since one of the purposes of the AT library is to be a pluggable component in many different applications, the API has been carefully defined. The API allows new GUI implementations as well as easy reuse of existing into new settings.

Furthermore the AT library is implemented in Java, since it is one of the most widely used programming language and it is more or less independent of operating system. The code can be run on the Java Virtual Machine that exists

2 <http://jena.sourceforge.net/>

3 <http://www.openrdf.org/>

4 <http://www.w3.org/DesignIssues/Notation3>

5 <http://www.w3.org/TR/rdf-syntax-grammar/>

for the most commonly used operating systems used today. The following subsections will describe how the design choices have been implemented. In order to understand the details you need to be familiar with the terminology presented in deliverable 3.2.

2.1 Design Overview

To make the annotation tool library useful in many different settings it has been made configurable in four ways. This is the same flexibility that the AT User Interface Integrators need to consider:

- The **source** of the metadata, ie where to load and save from. As well as which resources to edit metadata for.
- The **RDF library** to use for making changes to the metadata within the application.
- Which **Annotation Profiles** to use when editing, may depend on the context, i.e. user roles, access rights, editing purpose.
- Which **GUI** (Graphical User Interface) to use.

Hence, when using the Annotation Tool library these are the main aspects to consider for a successful adaption, and no other tool or framework, made for editing metadata, that cover this kind of flexibility has been found. An overview of the design of the Annotation Tool library is given in figure 2, the source of the metadata is the *Query Target*, the alternatives for RDF library are *Jena* and *Sesame*, the Annotation Profile is represented by the *Formlet* class⁶, and the GUI choices are here represented by the *MetadataPanel* and the early version of the LUISA annotation tool called *Eluisa*. In figure 2, the RDF library will be responsible for expressing RDF in several ways.

⁶ The formlet class may be renamed to Annotation Profile in the future.

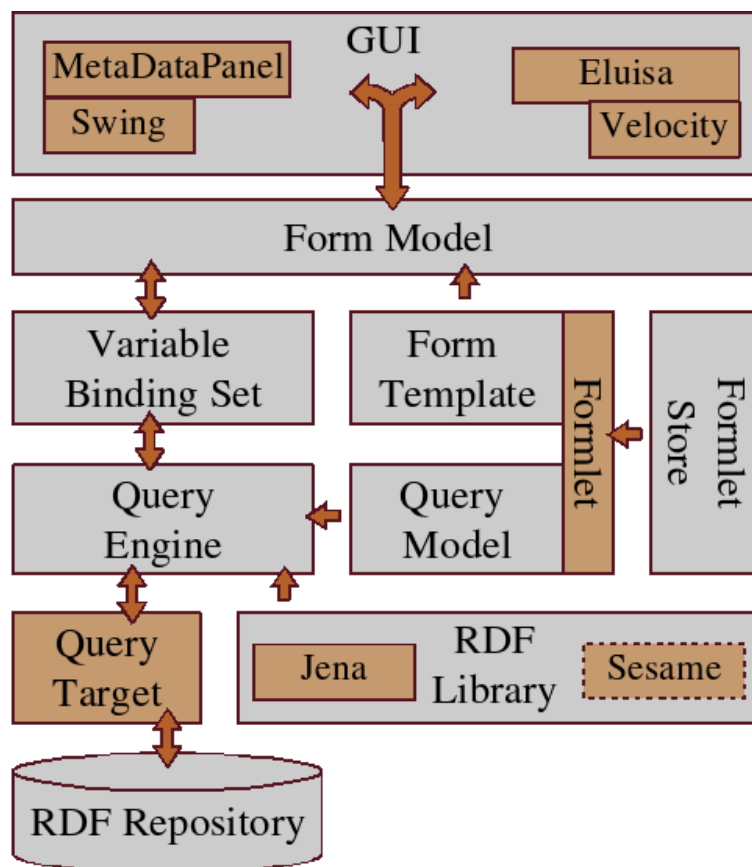


Figure 2: An overview of the main parts of the Annotation Tool Library, the parts colored in orange (Which are: Metadatapanel, Swing, Eluisa, Velocity, Formlet, QueryTarget, Jena and Sesame) are configurable.

In addition, the *FormModel* provides an API which is of interest for the AT user interface developers, since it provides access to the abstract model of the form that should be mirrored in the GUI.

The remaining grey boxes (not yet mentioned) in figure 2, i.e. *Template Form*, *Query Model*, *Query Engine*, and *Variable Binding Set*, all represent the inner workings of the Annotation Tool library and will be described more in the following sections.

2.2 Creating a form

How to create a form from an Annotation Profile and an RDF-model was covered in detail in Deliverable 3.2 - *Annotation Profile Specification*. This has been implemented in the Annotation Tool library and the important classes, and how they are related, is shown in figure 3. The name of the classes in that figure correspond to the terminology introduced in the section Glossary of terms in Deliverable 3.2, for example *Graph Pattern*, *Variable*



Binding, Form Template, Form Model, Form Item, etc. The class *QueryModel* introduced in figure 2 is a superclass of the *GraphPattern* class introduced here.

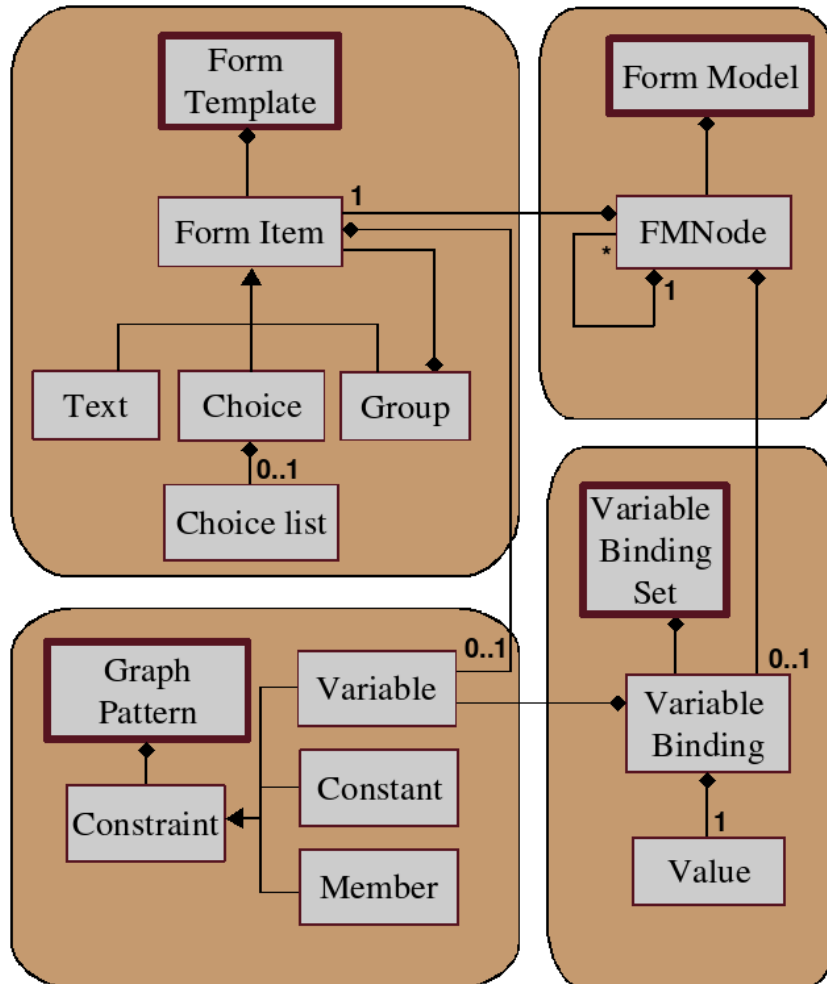


Figure 3: UML diagram showing the most important classes and their relations in the annotation tool library

An Annotation Profile consists of a Graph Pattern and a Form Template and before instances of their corresponding classes *Graph Pattern* and *Form Template* can be created they need to be read in from wherever they are stored, which usually is a file. Depending on how they are serialised different factory-classes are used to create the instances. It is also necessary to have an instance of the RDF-model to edit, but this is handled by the RDF library.

2.2.1 Creating the Variable Bindings

The Graph Pattern of an Annotation Profile defines what metadata structure that is possible to edit when using it. This is expressed in a query language such as SPARQL that needs to correspond to a tree-structure. The Graph Pattern when this is launched upon the RDF-model bindings are created from the

variables in the Graph Pattern to the corresponding node in the RDF model that defines what metadata to edit. The Graph Pattern is a tree structure T will be launched upon the RDF-model to create bindings from the variables in the Graph Pattern to a corresponding node in the RDF model. The variable bindings are used to reference where the changes in the RDF model will take place when the end user is editing in the created form.

The class Graph Pattern in figure 3 is not dependent on any specific query language and fulfills the design goal of being flexible on the syntax of the AP. Furthermore, the nodes in the RDF-model are abstracted in the Value class so that there is no direct dependencies to a specific RDF library. All variable bindings are collected in instances of the class Variable Binding Set and is created by the Query Engine. How the bindings are created is given in an example in figure 4, where a Graph Pattern is launched upon an RDF graph.

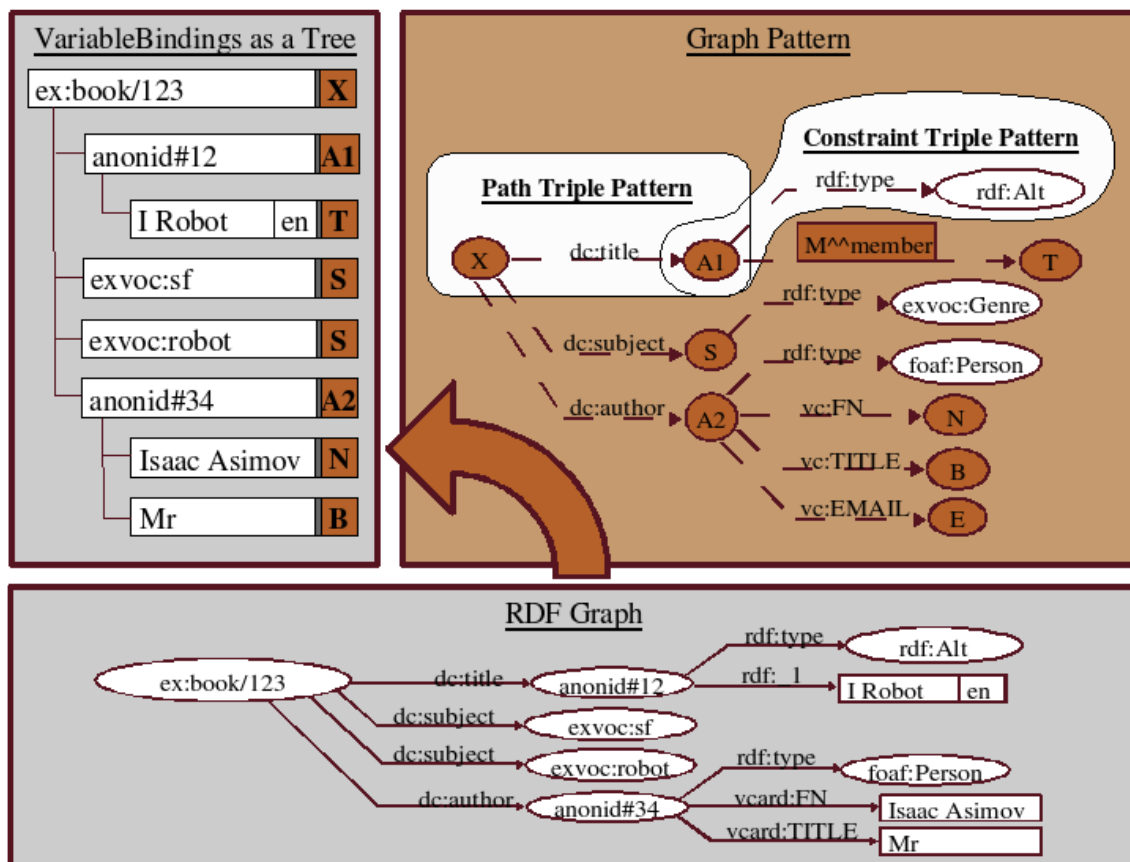


Figure 4: A Graph Pattern launched upon an RDF graph to create a set of Variable Bindings

2.2.2 Creating and using the Form Model

One of the design goal was to have a well defined Form Model API, from where a specific graphical user interfaces can be created. This is fulfilled in the class Form Model in the class diagram in figure 3. To construct a Form Model the Form Template of the Annotation Profile is combined with the Variable binding. The Form Template is the part of the Annotation Profile that defines how the metadata will be edited by outlining the structure of the form and expresses if value should be edited for example by using a free-text field or selected from a set of choices. The class Form Template is used for this purpose in the implementation and is like the Graph Pattern independent of serialisation and therefore also fulfills the goal of flexible use of AP syntax.

When implementing a particular user interface of the Annotation Tool, it is the class *-Form Model-* and its constituents that needs to be utilized. The *Form Model* consists of Form Model Nodes (called *FMNode*) that forms a tree-structure and represents the structure of the form to be displayed to the user.

A *Form Model* instance is created from a *Form Template* and a *Variable Binding Set*. A *Form Template* is also a tree-structure and consists of *Form Items* that can be of the types *Choice*, *Text*, or *Group Form Items*. If the type is *Group Form Item* they can contain yet another set of *Form Items* and this is used to create the tree-structure. Each *Form Item* contains a variable that should correspond to a variable that has a binding in the *Variable Binding Set*. Starting from the root of the *Form Template* the corresponding Variable binding is located in the *Variable Binding Set* (should also be the root) and an *FMNode* that will be the root of the *Form Model* is created. The *Form Template* tree is traversed and and *FMNodes* are created in a similar way where the rest of the *Form Items* are matched to a *Variable Binding*. Note that a an *FMNode* can only contain another *FMNode* if it consists of a *Group Form Item*. The procedure is illustrated in figure 5 below.

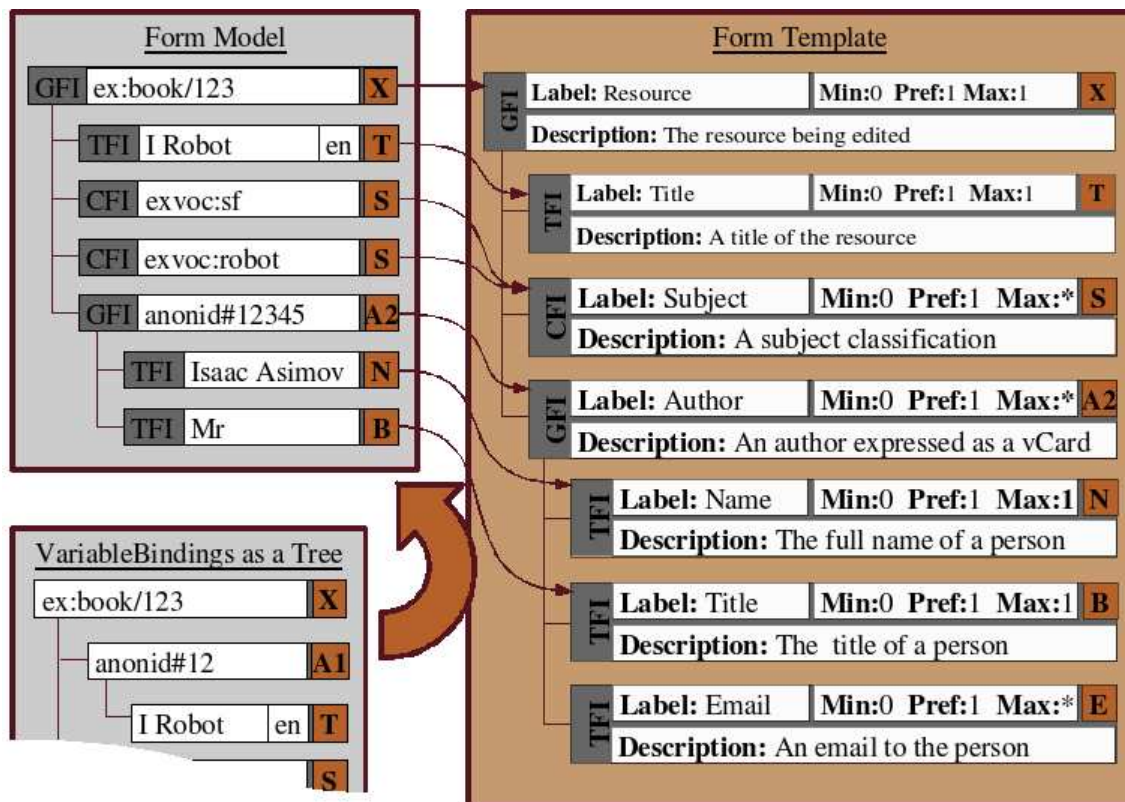


Figure 5: A Form Model is created from a Variable Binding Set and a Form Template

The *Form Model* contains the structure of the form to be displayed through how the *FMNodes* are structured. In order to create the form the tree-structure is traversed and depending on the *Form Item* a node contains a text-field or a choice should be rendered in the form. The node could be another grouping and then that should be rendered, for example by making an indentation.

2.2.3 Guidelines for Presenting Multiplicity

The *Form Item* also contains information on the multiplicity for the variable it has a reference to. With this information the application knows how many text-fields or choice items that it is allowed to create. Indications of what is possible to add or remove of the fields or items should be included in the interface and preferably with a '+' or '-' signs. The placement of these is of importance, since it tells the user how and where more values are added. For example when adding keywords to a resource, several keywords can be added in several languages. The signs where the keywords are expressed in more languages should be placed close to the languages-choice and to add another keyword the signs should be close to that label.

3 DESIGN AND ARCHITECTURE OF THE LUISA ANNOTATION TOOL

When constructing the LUISA Annotation Tool the advantages of the configurability possibilities in the Annotation Tool library was used by making the following choices:

- 1) The metadata **source** is located in a triple store that is accessed via an editing protocol.
- 2) For **RDF library** Jena was chosen.
- 3) For **Annotation Profiles** two LOM profiles was created according to the needs of the LUISA use-cases described in LUISA deliverable 3.5 [3].
- 4) For **GUI** we decided to build a rich web-application based on an established JavaScript toolkit and REST-ful principles.

The reason for accessing the source via a protocol was that the tool and the RDF-storage do not always run on the same machine, so therefore a protocol for accessing the repository had to be used. This actually opens up a possibility of using the same Annotation Tool to edit metadata on more than one repository. The tool is further distributed since it is accessed using a browser that communicates with a server running the Annotation Tool library with some specific adjustments made to it. The application in the browser is built by using JavaScript, and the environment of the web-browser was chosen since it is an environment that is known to most users and it makes the tool easily accessible where nothing has to be installed on the computer of the user. This leaves us with a setup of the LUISA Annotation Tool as depicted in figure 6 below.

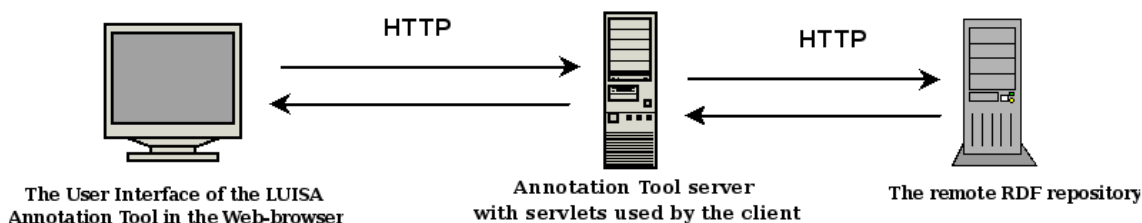


Figure 6: Distributed Annotation Tool

3.1 A typical editing scenario

The LUISA Annotation Tool is launched by pointing a regular web-browser to its location (a URL). After the tool has been initialised the user can choose which resource to edit from a list. Below are the steps taken for retrieving information for the chosen resource:

1. A request is sent to the server with the URI of the resource to edit together with another URI that identifies the Annotation Profile to use.
2. When the server receives the requests it loads the Annotation Profile and

sends a query to the remote RDF-storage to retrieve an RDF subgraph appropriate for editing using the AP given in the request.

3. With the subgraph and the Annotation Profile the server creates a Form Model and sends that together with the Form Template of the Annotation Profile back to the client.

The client renders the metadata for the chosen resource in a form. When the user interacts and changes the information, the changes are only done locally in the browser. To update the storage the user needs to actively save the changes. The following steps will take place when saving:

1. The Form Model is sent to the server, together with a pointer to an Annotation Profile that was used in retrieval step. See discussion above.
2. The old subgraph is modified by binding it again to a Form Model, as in step 3 above, and then merging with the modified Form Model received from the client. This ensures that no information in the subgraph that was not bound into the Form Model is lost. The now modified subgraph together with the query from step 2 above is sent as an update request to the remote storage.
3. The remote storage can from the query calculate the subgraph that has been changed. That whole subgraph is removed and the subgraph sent with the update request is inserted instead. After a complete and successful operation the remote storage sends a notification that tells the server if the update went OK. If not, an error-message is sent instead.
4. The server in turn sends this notification further to the tool running in the client. If something has failed during the process the user is notified.

3.2 Techniques used for the GUI

As web 2.0 technologies have matured, gained in popularity, diversified, and available toolboxes have become more professional there is a strong tendency towards building GUIs as rich web-applications utilizing JavaScript toolkits. For the LUISA Annotation Tool there are at least four strong reasons for building a web-application with the help of a JavaScript toolkit:

- 1) easy integration into Learning Management Systems that are often web-based,
- 2) appropriate separation between Annotation Tool Library and the GUI,
- 3) a JavaScript toolkit provides a range of ready made components, widgets, suitable for form creation,
- 4) localization and accessibility is already provided for.

For reasons of speed and simplicity we chose to use JavaScript Object Notation

(JSON⁷) for sending data between server side Annotation Tool library and the JavaScript based client in the browser. In theory, the server side support could be avoided by having a direct connection to the triple store. However, this would require a total reimplementaion of the Annotation Tool library in JavaScript as well as a mature and stable JavaScript based RDF library. As this is not feasible we continued by deploying the Annotation Tool library on the server side and established a light-way protocol for server to client communication. The simplest and cleanest approach is called REST [4], which basically recommends a simple use of various HTTP verbs (mainly GET and POST) along with parameters.

3.3 The interface to the end-user

When the LUISA Annotation Tool is launched in the browser, a user interface similar to the screenshot in figure 7 is displayed to the user.

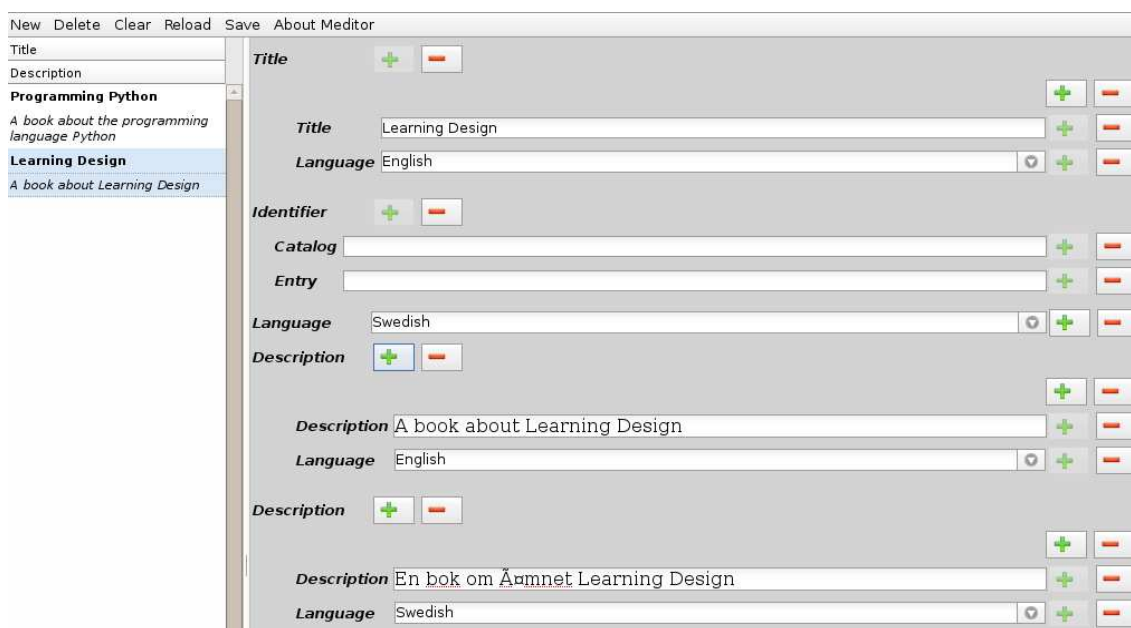


Figure 7: The interface rendered to the end-user

As can be seen in that screenshot of figure 7, the tool is divided into three parts, a form to fill in metadata to the right of the screen, a list of available resources to the left, and on the top a toolbar. Selecting a resource on the list to the left will open up a form displaying the corresponding metadata record. The buttons of the toolbar stays the same, but are inactivated when they cannot be used.

By using JavaScript and AJAX⁸ to build the LUISA Annotation Tool user interaction does not lead to entire page reloads. Instead only parts of the web-

7 <http://www.json.org>

8 <http://dmoz.org/Computers/Programming/Languages/JavaScript/AJAX>

page is updated when information is available. For example when the list of resources is updated a request is sent to the server which returns information only to update the list which is reloaded. This leads to an update of the list to the left, whereas the rest of the GUI stays intact.

The toolbar is mainly used to manage the metadata record for a selected resource. For example to delete, create new, and save changes made on a metadata record. By clicking on the button

- **New**, a new metadata record for a resource will be created. Observe, that the record is not stored in the repository until the save-button has been clicked,
- **Delete**, will delete the current displayed metadata record,
- **Clear**, will clear the current form of all field values,
- **Save**, the current metadata displayed in the form will be saved with the changes made to it,
- **Reload**, will update the list of resources displayed in the left side of the screen.

A more detailed description of how to interact with the tool can be found in LUISA deliverable 3.4 – *Annotation Tool Documentation* [5].

3.4 Communication between the client and the server

- The server-side support consists of a set of services that is accessible over HTTP via GET and POST commands in accordance with the principles of REST, where each service accepts a list of parameters. To simplify for the The LUISA Annotation Tool, communication with the services is done via the JSON data transport. In order to support the GUI presented above, the services must be able to handle the following operations:
 - Retrieve a list of resources,
 - Create a new metadata record for a resource
 - Retrieve a form template, Retrieve a form model for a particular resource,
 - Save changes done to a form model,
 - Delete the metadata record of a resource.

In the implementation of the LUISA Annotation Tool the support to load, save and delete metadata record of a resource are handled by a single Java Servlet⁹, but with different parameters for each operation. The support to retrieve a list of resources and to create a new metadata record for a resource are done by calling separate servlets.

⁹ <http://java.sun.com/products/servlet/>

3.4.1 Retrieve a list of resources

When the tool creates the list of resources possible to annotate it needs to request them from the remote repository. The simplest way is to send a query directly to the remote repository, but for security reasons JavaScript is only allowed to send request to the same server as the page was loaded from. So, the main task of the servlet is to act as a proxy and forward the query to the remote repository. The parameters sent to the servlet is described in the table 1 below:

Table 1: Parameters

Servlet name:	/sparql
Description:	This servlet is used as a proxy to query a remote RDF-repository that has a SPARQL endpoint. Hence, only GET-operations are possible. For the LUISA Annotation Tool the servlet is used for querying the set of Learning Resources possible to edit and to present them in the list inside the tool.
Parameters:	<p>sparql, the value of this parameter is a valid SPARQL SELECT-query. Other queries will not be accepted and an error will be returned..</p> <p>repository, the URL of the repository to forward the question to.</p>
Example URL:	<pre>http://www.example.com/sparql?sparql=PREFIX%20rdf%3A%20%3Chttp%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-ns%23%3E%20PREFIX%20lom%3A%20%3Chttp%3A%2F%2Fwww.cc.uah.es%2Fie%2Font%2Flom2wsml%23%3E%20PREFIX%20eads%3A%20%3Chttp%3A%2F%2Feads.org%2Ftraining%23%3E%20SELECT%20%3Fres%20%3Ftitle%20%3Fdesc%20WHERE%20%7B%3Fres%20rdf%3Atype%20eads%3AeadsTraining.%20OPTIONAL%7B%3Fres%20lom%3Atitle%20%3Fy.%20%3Fy%20lom%3AhasSingleLangString%20%3Fz.%20%3Fz%20lom%3AhasCharacterString%20%3Ftitle%20OPTIONAL%7B%3Fres%20lom%3AdescriptionLO%20%3Fy2.%20%3Fy2%20lom%3AhasSingleLangString%20%3Fz2.%20%3Fz2%20lom%3AhasCharacterString%20%3Fdesc%20%7D%7D%20&repository=http%3A//localhost%3A8080/joseki/books</pre>
Returns:	The resulting result-set of the query serialised in JSON.

For the LUISA Annotation Tool the list of possible resources to edit contains Learning Objects, and they are listed with their title and description. For that purpose a hard-coded query could have existed on the servlet, so that the only parameter sent is the repository to query. It is however not always the case that only the title and description of the LO needs to be presented in the list, so in order to support easy modifications the servlet was built to support any SELECT SPARQL-queries.

3.4.2 Create a new metadata record for a resource

In order to create a new metadata record for a resource, a new unique URI is created that maintains the relation between the resource and the metadata record. Depending on the requirements of how a new URI should be created and the importance of uniqueness this could have been fully implemented on the client side. But with a call to a servlet it is possible to verify the uniqueness of the URI. How to call the servlet is described below, in table 2:

Table 2: How to call the servlet

Servlet name:	/newuri
Description:	Called when an new URI should be created.
Parameters:	baseURI , the base-URI where upon a new unique URI has to be created. remoteServiceURL , an optional parameter that indicates where to look for the uniqueness of the URI
Example URL:	http://www.example.com/newuri?baseURI=http%3A//www.uhp-nancy.fr/ontologies/LUISA%23
Returns:	A new unique URI.

3.4.3 Retrieve a form template

The form template is needed for the values in the choice-items, since this is not included in the Form Model. The form templates rarely changes and can therefore be cached on the client-side, so a call to the servlet is not necessary if a cache instance of the template exist. This means that when a template is to be loaded the client first looks if this template is in the cache, if not the servlet with the Form Templates is called. This servlet is described in the table 3.

Table 3: Servlet

Servlet name	/template
Description:	Called when the client needs to load a Form Template, hence only GET operations are possible. The templates does not change very often and a loaded template can therefore be cached on the client-side. When a template has been loaded once from the server it will also be cached there in order to improve performance.
Parameters:	ap , the value given here is a URI that identifies an Annotation Profile
Returns:	If the template can be found and loaded it is returned to the client in a JSON-format.
Example URL:	http://www.example.com/ template ? ap =http%3A//kmr.nada.kth.se/shame/Formulator/formlet%23UHP-all

Retrieve, save and delete metadata records This servlet is responsible for managing the form model, both retrieving it and pushing it back when changes has been made. The HTTP GET operation is used to retrieve and HTTP POST is used to push changes back (Save). In addition, pushing back an empty form model will lead to an empty metadata record, which results in an empty metadata record, hence a separate solution for delete is not necessary. The servlet is described in table 4:

Table 4: Servlet (II)

Servlet name:	/form
Description:	This servlet can be called using both GET and POST. GET-action: a Form Model serialized in JSON will be returned. Used when the Form Model should be rendered in the interface. POST-action: together with the request a Form Model (serialized in JSON) is sent. This is used when changes done to the Form Model in the client should be updated.
Parameters:	resource , the URI that identifies the resource to edit sparqlservice , the URL to the remote storage from which the subgraph to edit can be retrieved.

	ap , a URI that identifies the Annotation Profiles to use.
Returns:	GET-action , returns a Form Model serialised in JSON POST-action , returns a confirmation if the changes in the form was carried out successfully.

3.5 The server side and communication to the remote repository

RDF is used as the standardized way of representing data on the Semantic Web, but rather surprisingly no standardized way or protocol was found to modify RDF on a remote basis. Early initiatives suggested in [6] and [7] have not gain wide acceptance which is probably due to the lack of a common query language. With SPARQL becoming a W3C recommendation a standardized query language exist for RDF. Upon this standard an approach to update/modify RDF on a remote basis has been developed under the name SPARUL [8]. This approach seemed promising at first, but from a perspective of building an Annotation Tool upon, the approach falls a bit short. This is mainly because a lot of knowledge of the RDF graph on the remote storage is required in advance. To use SPARUL would be possible, but would also be unnecessary complicated.

The details about why SPARUL is not the best choice is described further in [9], where the approach chosen for the LUISA Annotation Tool is also described. The basic idea is to restrict the remote editing to only involve a subgraph of the whole model at the RDF-repository. So, when a user has chosen a resource to edit (from the list in the tool) the servlet described in section 0 performs a two step procedure:

1. Fetch the subgraph to edit from the remote storage by sending a query.
2. After the subgraph has been edited/changed and requested to be saved the servlet will first remove the initial subgraph retrieved in step 1 and after that insert the modified subgraph.

To realize this SPARQL was used as the query language, by using queries of the type called DESCRIBE. The result of such a query is an RDF-subgraph, but unfortunately is DESCRIBE not formally defined and leaves it up to the implementor to decide exactly what to return. This means that different implementation might return different subgraphs, however the normal approach that goes under the name anonymous closure can be described as:

- Include all direct statements of all matching of variables and the URIs given in the request,
- If an anonymous node appears in those statement add all the direct statement for that node as well. If new anonymous nodes are found keep

repeating until no anonymous node appears as a leaf in the graph.

3.5.1 Constructing a query to retrieve appropriate subgraph

When the form-servlet is requested to deliver a form model for a resource, it must first retrieve a sufficiently big subgraph by querying the remote repository. The query needs to be carefully designed in order to not result in a subgraph that contains too little or too much information.

A DESCRIBE query that only includes the URI of the resource to edit would return the anonymous closure of that resource. In many cases this would be enough, but if the metadata record has intermediate resources that are not anonymous their further properties will not be included. A better alternative is to build a DESCRIBE query from the Graph Pattern of the Annotation Profile. A Graph Pattern is a tree-structure that defines what metadata to edit, and is expressed in a query-language. Now a query constructed using the Graph Pattern and just replace the root with the resource to edit would instead result in a model that contains too much information. The variable-leaves, ie the variables that does not have any outgoing properties leading to other variables, will be matched and their anonymous closure included. However, the anonymous closure of these resources contains statements that cannot be matched by a variable binding with the Graph Pattern when the form model is created. Therefore, the variable-leaves can be removed from the query without losing information in the resulting subgraph that is relevant for the creation of the form model.

So, the query that returns the appropriate model is constructed with the help from the Graph Pattern by replacing the root with the resource and remove all variable-leaves. That pattern has to be converted into SPARQL and all the statements expressed in the query have also be turned into OPTIONAL, since we want a model returned even if a variable have no matches. In the table below is an example from an Annotation Profile that is constructed to be used by the industrial use-case in used in LUISA. The intent of this profile is to edit the title (luisa:title) of a resource of type eadsTrainingis that is a specialisation of shown in the table below:a Learning Object.

Table 5: Example from an Annotation Profile in LUISA

<pre>SELECT * WHERE { ?X rdf:type eads:eadsTraining . ?X luisa:title ?Y . ?Y rdf:type luisa:langString . ?Y luisa:hasSingleLangString ?LSS . ?LSS rdf:type luisa:langStringSingle . ?LSS luisa:hasCharacterString, ?T .</pre>	<pre>DESCRIBE <http://www.example.com/lo1> ?Y ?LSS WHERE { OPTIONAL { http://www.example.com/lo1 luisa:title ?Y. } OPTIONAL { ?Y luisa:hasSingleLangString ?LSS .</pre>
---	---

the repository is described in the table 6. The servlet used to query the repository is a bit more complex, but this is documented at www.joseki.org.

Table 6: Example from an Annotation Profile in LUISA

Servlet name:	/update
Description:	Called when the repository should be updated. A call to GET or POST will perform the same action.
Parameters:	remove , this is a SPARQL DESCRIBE query from which the subgraph to remove can be calculated.
	insert , an RDF-model serialised into RDF/XML. This is the model to be inserted into the model on the repository.
Returns:	An XML-snippet informing if the update was successful or not.

When the repository receives the request, the model to remove is calculated by querying the main model on the repository. The resulting subgraph from that query is removed from the repository. The updated RDF subgraph to be read in from it serialisation and then put into the main model. After this is done the repository has been properly updated.

3.6 Using the Annotation Tool to display metadata

Apart from editing metadata, the Annotation Tool library can be used also to create an interface where the metadata is only displayed. This is done in the same way as when a form is created, as has been described in this deliverable. But instead of making the values possible to edit the user interface is implemented to only display the values.

This possibility has been used in LUISA when information on a Learning Resource is requested by the end-user. For example, when an end-user has made a search for suitable Learning Objects to fill a certain competency gap only some parts of the metadata is displayed. In order to see the rest of the metadata the an interface using parts of the LUISA Annotation Tool has been used that displays all the metadata for the requested Learning Object.

4 ENCOUNTERED PROBLEMS

The LUISA Annotation Tool is designed to edit RDF, this means that the metadata has to be translated into WSMML. This is the language the ontologies are stored in and is used by the rest of the architecture. This translation is not performed by the LUISA Annotation Tool, but takes place in the rest of the architecture by using a translator. This translation works well in most cases, since only instances in the ontologies are annotated. However, some problems were encountered and are described below.

4.1 Datatypes

Datatypes are used in RDF to represent if values should be integers, strings, dates and so on. The same approach exist in WSMML to represent different kinds of values. The problem with here is that the official mapping from WSMML to RDF [ref?] does not define how datatypes should be handled and is a constraint of the current implementation of WSMO.is that it was not supported by the translation from RDF to WSMML and the other way around it was simply ignored. The Annotation Tool Library has been designed to make use of information about datatypes since it can be expressed in the Annotation Profile. From that information different ways of interacting with the form can be presented by the user depending on the datatype, for example if the datatype is integer the value can be edited in the form with a slider which can be slided back and forth in order to edit the value. Unfortunately, due to that no official mapping exist for datatypes these additional ways interacting derived from datatypes cannot be used in the LUISA Annotation Tool.

4.2 Language typed literals

Sometimes natural language is used as values in metadata. To avoid guessing, it is useful to specify which language used together with the string value. The same value could even be provided in multiple languages, this is often the case for example titles and the descriptions of Learning Resources. The approach of RDF is to allow language tags on literal values for telling the language of the literal. This is a simple and standardised way to express the language of a literal. In the Annotation Profile a special configuration in the Form template on a Text Form Item results in a language chooser on the right of the string input field.

Unfortunately, there is no built in support in WSMML for expressing language tags on literals (which in WSMML is called Strings). The workaround used in the LUISA ontologies is to not point to a WSMML string directly but to point to an intermediate instance which has two attributes, a string and a language. Unfortunately this leads to additional complexity in the LUISA ontology and Annotation Profile as well as a less smooth user interface due to not being able to use the built in support for language tags in the Annotation Tool GUI.

4.3 Additional information for translating to WSML

The instances of the ontologies in WSML have to belong to an ontology and when the instances in WSML are translated to RDF information is added that tells to which ontology a certain instance belong. If an instance is created in RDF, information about which ontology it belongs to needs to be added in order to translate to WSML. This problem was solved by applying a filter in the update procedure of the Joseki-server. If the instance already exists in the repository, the additional information does not need to be added. If it does not exist, the instance is assumed to be new and this information is added. To which ontology it belongs to is decided by the type of the instance.

5 CONCLUSION

This report has described the Annotation Tool from the technical perspective. For the role of the *Code Library Developer* an overview of how the Annotation Tool Library produces a Form Model from an Annotation Profile was given. An overview of the Form Model was also given as information to *User Interface Developer* that develops specific interface of the Annotation Tool. For the *User Interface Integrator* the specific integration done for LUISA was presented.

The specific LUISA Annotation Tool will not always run on the same machine as where the RDF is stored. Therefore a way to update the RDF repository on a remote basis via a protocol was designed and implemented. When editing this way an appropriate subgraph is retrieved from the repository by sending a query. The tool will then create a form where the user can manipulate the metadata. When the user saves the changes made to the form two things are sent to the repository, 1) the updated RDF subgraph and 2) the original query used to retrieve the subgraph. The query is used to calculate the subgraph to remove and after that the updated RDF subgraph is inserted. In order to retrieve the appropriate subgraph the query could be easily derived from the Graph Pattern of the Annotation Profile.

Many applications nowadays are required to work over the World Wide Web and should also be easy to integrate (eg mash-ups) into other web application, all in the spirit of Web2.0. Therefore it was chosen to develop the user interface of the LUISA Annotation Tool in JavaScript using the principles of REST. That way the tool can be loaded by using a web-browser, which is a common environment for most computer users today. The tool can also easily be integrated into other web-environments such as a Learning Management System. The LUISA Annotation Tool requires the functionalities of the Annotation Tool library to be available on the server side as it is not realistic to duplicate them in JavaScript. The communication to the server is done in a REST-ful manner and information is sent in the JSON format .

Unfortunately the mapping from RDF to WSML makes it impossible to use the datatype information in the Annotation Profiles. The way datatypes are used in both RDF and WSML seems similar and we believe a mapping should be possible to make. Since WSML is built on an ontology model additional information needs to be provided when an instance to an ontology is created in RDF. Otherwise the translation will not work as wanted. When the LUISA Annotation Tool creates a new instance the additional information is performed in the update procedure of the remote repository. Another more critical issue was that the strings in WSML could not be tagged with a language in a simple way. The workaround used in LUISA leads to both additional complexity and an unnecessary crude form. We believe that if WSML provided built in support for language tags, it would be one obstacle less for gaining a wider acceptance on the WWW.

REFERENCES

- [1] Palmer, M.; Enoksson, F.; Nilsson, M.; Naeve, A. (2007) *Deliverable 3.2 Annotation Profile Specification*, Deliverable in the LUISA project, Available at <http://www.luisa-project.eu>
- [2] Enoksson, F.; Palmér, M.; Naeve, A.; Arroyo, S.; Fuschi, D.; Pariente, T. (2007) *Deliverable 3.1 State of the art, SWS Infrastructure, Annotation, LCMS*, Deliverable in the LUISA project, available at <http://www.luisa-project.eu/>
- [3] Grandbastien, M.; Huynh Kim Bang, B; Monceaux, A. (2007) *Deliverable 3.5v1 Use-case annotation profiles*, Deliverable in the LUISA project,
- [4] Fielding, R. T. (2000): *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation at the University of California, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [5] Enoksson, F.; Palmér, M.; Naeve, A.; Nilsson, M. (2007) *Deliverable 3.4 Annotation Tool Documentation*, Deliverable in the LUISA project, available at <http://www.luisa-project.eu>
- [6] Nejdli, W.; Siberski, W.; Simon, B.; Tane, J. (2002): *Towards a Modification Exchange Language for Distributed RDF Repositories.*, Proceedings of the First International Semantic Web Conference on The Semantic Web. Springer, UK
- [7] Seaborne, A. (2002): *An RDF NetAPI* , Proceedings of the First International Semantic Web Conference on The Semantic Web. Springer, UK
- [8] Seaborne, A.; Manjunath, G. (2007): *SPARQL/Update A language for updating RDF graphs. Version 2.* , <http://jena.hpl.hp.com/~afs/SPARQL-Update.html>
- [9] Enoksson, F.; Palmér, M.; Naeve, A. (2007): *An RDF Modification Protocol, based on the Needs of Editing Tools*, Proceedings of the 2nd International Conference on Metadata and Semantics Research (CD-ROM), Ionian Academy, Corfu, Greece <http://www.mtsr.ionio.gr/proceedings/enoksson.pdf>