

LUISA

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

IST- FP6 - 027149



Deliverable D3.6

Documentation on the use of Annotation Profiles

Fredrik Enoksson
Matthias Palmér
Ambjörn Naeve
Benjamin Huynh Kim Bang

Due date of deliverable: 31/05/2008

Actual submission date: 30/06/2008

Start date of the project: 01/03/2006

Duration: 30 Months

Fredrik Enoksson

ULL

Version 1.0, dated 20/06/2008

Change History

Version	Date	Status	Author (Partner)	Description
0.1	20080519	draft	Matthias Palmér (ULL), Fredrik Enoksson (ULL), Ambjörn Naeve (ULL)	Outline of what to contain in this deliverable
0.2	20080522	draft	Fredrik Enoksson (ULL)	Several chapters written and in draft status, the rest are outlined with the main idea/purpose for what they should contain
0.3	20080529	draft	Benjamin Hyunh Kim Bang (ULL)	The chapters dealing with the end-user experience was written.
0.4	20080530	draft	Matthias Palmér (ULL)	Chapters containing ideas on a Annotation Profile written
0.5	20080611	draft	Fredrik Enoksson (ULL), Ambjörn Naeve (ULL)	Reworking all the chapters.
0.6	20080620	draft	Ambjörn Naeve (ULL), Fredrik Enoksson (ULL)	Changes done according to the internal review and check of spelling and language.
1.0	20080620	Final	Fredrik Enoksson (ULL)	Final version
1.0	20080623	Final	Mónica Miró (ATOS)	Some minor format changes

EXECUTIVE SUMMARY

In LUISA the mission is to investigate and exploit a Semantic Web Service Architecture to work in the context of Learning Management Systems and Learning Object Repositories. By describing learning objects with semantically clear metadata, these systems will be able to talk to each other and efficiently and effectively process learning objects on behalf of users.

In LUISA the creation and editing of learning object metadata records are handled by a configurable Annotation Tool. The configuration mechanism used is based on Annotation Profiles which controls both which metadata structures to edit as well as the form-like appearance that the end-user will see. In this report we list some advantages and drawbacks, of using Annotation Profiles as well as some possible future directions.

We have found that annotation profiles simplify the creation and modification of editor software for deep and complex metadata structures. This has been useful in LUISA since we have iteratively developed several large ontologies with many complex relations. In addition, due to the flexibility of Annotation Profiles, we have been able to test and deliver several editors tailored for different settings and authoring roles.

As expected, the ontologies used in LUISA do not (and in general should not) contain enough information to act as Annotation Profiles in themselves. However, the ontologies act as a base layer on top of which Annotation Profiles provide order, labels, layout and authoring specific interaction instructions etc. In addition, experiences from creating Annotation Profiles via a specific tool are discussed as well as an outline of a new improved Web-based version of the tool.

Document Information

IST Project Number	FP6 – 027149	Acronym	LUISA
Full title	Learning Content Management System Using Innovative Semantic Web Services Architecture		
Project URL			
Document URL			
EU Project officer	Francesco Barbato		

Deliverable	Number	D3.6	Title	Documentation on the use of Annotation Profiles
Work package	Number	3	Title	Learning Object Annotation

Date of delivery	Contractual	31/05/2008	Actual	30/06/2008
Status	Version 1.0, dated 20/06/2008		final <input checked="" type="checkbox"/>	
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)	For the Semantic Web Service architecture developed in LUISA to prove useful the resources involved, mainly Learning Objects, need to be described with semantically clear metadata. To edit metadata on these resources an Annotation Tool has been developed that is configurable with a mechanism called Annotation Profile. These profiles define what metadata that can be edited and also outline the resulting form presented to the end-user. The experiences from creating and using such profiles is described in this document, together with some ideas on how they can be improved.
Keywords	Metadata Annotation Profiles

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 Interactive Labs S.r.l.	GIUNTI 	Dr. Fabrizio Giorgini Giunti Interactive 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 – Corporate Research Centre		Anne Monceaux EADS France- Innovation Works. 18, Rue Marius Terce – BP 13050 – 31025 Toulouse Cedex 03, France. Email: anne.monceaux@eads.net Tel.: +33 561-184-725 Fax: +33 561-187-611

TABLE OF CONTENTS

EXECUTIVE SUMMARY	3
TABLE OF CONTENTS	6
1 INTRODUCTION	7
1.1 List of abbreviations used in this document	7
2 EXPERIENCES FROM BUILDING THE USE CASE ANNOTATION PROFILES	7
2.1 Keeping Annotation Profiles synchronized with ontologies	8
2.2 Editor scope does not follow from the ontology	11
2.2.1 An example from a LUISA ontology	11
2.3 Using Annotation Profile Snippets – Formlets.....	13
2.4 Using the Formulator tool.....	13
3 IDEAS FOR IMPROVING THE MANAGEMENT OF ANNOTATION PROFILES	15
3.1 A Simplified Annotation Profile format	15
3.2 Improved Annotation Profile management.....	16
3.3 Creating skeleton Annotation Profiles from ontologies	18
3.4 Validating Annotation Profiles against ontologies	19
4 EXPERIENCES FROM EDITING (AND IDEAS FOR IMPROVING)	19
4.1 Complexity of deep data-structures	19
4.2 Use of controlled vocabularies.....	20
4.3 Need for check of constraints and for required values.....	21
4.4 Need for default values	22
4.4.1 Automatic creation of values	22
4.5 Connecting Learning Objects.....	22
4.6 To edit many resources in a repository	22
5 IDEAS FOR IMPROVING THE EDITING EXPERIENCE	23
5.1 Extending the Annotation Profile.....	23
6 CONCLUSION	24
REFERENCES	25

1 INTRODUCTION

The LUISA project aims at investigating (and exploiting) how a Semantic Web Service Architecture can work in the context of Learning Management Systems and Learning Object Repositories. The main resources involved are so called Learning Objects that will be matched to a Learning need given from the end-user. In order to get a match, the resources need to be annotated with structured data, ie metadata.

For the purpose of annotating metadata on these resources an Annotation Tool has been developed in work package 3 of the project. This tool is configurable with respect to what metadata that can be edited and also to some extent how it will be annotated. The configuration mechanism is called Annotation Profiles and were described in LUISA deliverable 3.2 *Annotation Profile Specification* [1] and also in this paper [2]. With the introduction of Annotation Profiles, work that was previously largely the responsibility of developers has now been specialized and divided up among several roles. For instance, the responsibility to create Annotation Profiles requires general knowledge of metadata standards as well as specific knowledge of the domain area. The responsibility for this is taken by a role we have chosen to call *Annotation Profile author*. It is also beneficial for this role to have good connections with the users of the metadata editor as well as knowledge about user interface design.

The experiences during the LUISA project of the role as an Annotation Profile author are covered in section 2 where a tool to support the construction and administration of Annotation Profiles is presented. This is followed up by ideas on how to further simplify this task in Chapter 3. Here the ideas to how a new version of this tool should look like is outlined. In chapter 4 some problems are presented that relates to the editing experience when using Annotation Profiles and in the following chapter 5 some ideas are presented to how the problems of chapter 4 can be tackled.

1.1 List of 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

WSML – Web Service Modelling Language

2 EXPERIENCES FROM BUILDING THE USE CASE ANNOTATION PROFILES

The purpose of introducing Annotation Profiles is to simplify the development of standards-compliant metadata editors. Whether it is worth the effort to use

Annotation Profiles depends on the complexity of the metadata constructs. If the aim is to edit only a few direct properties, Annotation Profiles are probably not necessary. In this case it is easier to simply build a custom metadata editor. However, if you

- have cardinality restrictions on your description fields,
- are using controlled vocabularies for your field-values,
- require validation according to datatypes, or
- just have many or deeply nested properties,
- Annotation Profiles can be really useful.

In the LUISA project, the metadata editors are both large and complex with restrictions according to defined ontologies as well as cardinality constraints. Therefore Annotation Profiles are used and during the development process we have experienced both flexibility and usefulness as well as limitations and time-consuming steps in the development and use of them. Solutions and workarounds to these issues will be discussed below.

2.1 Keeping Annotation Profiles synchronized with ontologies

Annotation Profiles have proved to be a flexible way to update a metadata editor according to an external specification. The Annotation Profiles were specified in the LUISA deliverable 3.2v2 [1], but they are also briefly reviewed here for the convenience of the reader. The purpose of an Annotation Profile (AP) is to act as a configuration mechanism for metadata editors. APs are descriptions outside any vocabulary or ontology, but uses information from such structures whenever possible. An Annotation Profile consists of two parts that are called:

- *Graph Pattern*, that defines the metadata structure that can be edited, and which can be expressed using a query language.
- *Form Template*, which outlines the form that the end-user will use to edit the metadata, and which contains information about cardinality restrictions and defines where in the structure of the Graph Pattern the end-user can edit the metadata. This part can for example be expressed in RDF and XML.

In LUISA a set of Annotation Profiles have been developed to edit instances in the LUISA ontologies. These ontologies are created using WSML (Web Services Modeling Language) and in the rest of this document we will therefore use the terms *concepts*, *instances* and *attributes*, which are part of WSML. A simplified example of a concept with one attribute - taken from an ontology used in LUISA - is presented below:

```
concept learningObject
  status impliesType (0 1) lomStatusVocabularyItem
```

Listing 1: An example of a structure in a LUISA ontology

In this example the concept *learningObject* has been stripped to only contain one attribute called *status*. Furthermore, the value of this attribute must be an instance of (sub-)type *lomStatusVocabularyItem*. For this structure we know that the value of the attribute 'status' will be edited in an editor, i.e. we know that we do not want to edit something in the structure of the concept *lomStatusVocabularyItem*. In an editor, this value will probably be edited using a dropdown menu that contains all the instances of this concept. So, when constructing an Annotation Profile to edit the value of *status*, the Graph Pattern presented in listing 2 below (in a SPARQL-like syntax) can be used.

```
SELECT * WHERE{
  X rdf:type luisa:learningObject.
  Y rdf:type luisa:lomStatusVocabularyItem.
  X luisa:status Y.
}
```

Listing 2: Graph Pattern used to edit the value of the attribute status

In the Form Template of this Annotation Profile we will need to indicate that the variable Y corresponds to the place in the structure that should be edited. We also need to indicate that the values will be chosen from a set of predefined choices (for example a drop-down menu). This is done in the Form Template given in listing 3 below, and - as can be seen there - the variable Y is inside a so called Choice-form item, which means that the value of Y will be chosen from a predefined set of choices. This set is populated by finding the instances that conform to the restriction of the variable Y in the Graph Pattern, i.e. all instances of *lomStatusVocabularyItem* in the underlying ontology.

```
<rdf:RDF
  <FT:Form>
    <j.0:query rdf:resource="http://example.org/luisa/status" />
    <rdf:li>
      <FT:GroupFormItem>
        <j.0:variable rdf:resource="http://example.com/luisa/var#X"/>
        <rdf:li>

          <FT:ChoiceFormItem>
            <dc:title>
              <rdf:li xml:lang="en">2.2 Status</rdf:li>
            </dc:title>
            <FT:variable rdf:resource="http://example.com/luisa/var#Y"/>
            </j.0:ChoiceFormItem>

          </rdf:li>
        </j.0:GroupFormItem>
      </rdf:li>
    </j.0:Form>
  </rdf:RDF>
```

Listing 3: Form Template used to edit the value of attribute status

From this example one can easily see that changes in the ontology will force a change in some of the Annotation Profiles. In the LUISA project, the main ontologies (i.e. the ontologies used by both use-case partners) have been rather stable, but some changes in attribute- and concept names have been carried out. The use-case specific ontologies have gone through more changes during the project, and the Annotation Profiles for them were therefore created later on in the project. As the metadata constructs in LUISA are quite complex, it would be rather difficult task to have to change the code of the Annotation Tool and recompile it for every change in the ontology. Instead, time could be saved by just changing the corresponding Annotation Profiles instead. Moreover, the APs can be expressed formally (i.e. in a syntax that can be parsed and 'understood' by a machine). Therefore it is possible to write a program that evaluates if the AP can be used to edit metadata-structures according to the ontology. This would be very hard - if not impossible - without the AP approach, since that would mean interpreting the code of such an Annotation Tool.

2.2 Editor scope does not follow from the ontology

If an editing tool could be created directly from an ontology, the Annotation Profile approach would have been unnecessary, since the APs would be given directly from the ontologies. In the example above one could argue that the Graph Pattern part could be derived from the ontology, which is true in this case that has a very simple structure with only one direct attribute. However, more complicated cases can be easily found and an example of that will be given below. For the Form Template of the example above one might also argue that it can be derived from the ontology, but this would entail a couple of assumptions. One such assumption is that we want to edit the value of the direct attribute *status*, which happens to be true in this case. But consider the case where *lomStatusVocabularyItem* has attributes that we would like to edit (starting from *learningObject*). Hence it is not always possible to automatically detect where in the structure the end-user wants perform edit operations.

2.2.1 An example from a LUISA ontology

Under some extra assumptions it would actually be possible to create an editor directly from the ontology, but for the resulting editor to be useful, a rather simple structure of the ontology would be required. Below an example is given where it is impossible to create a useful editor with only information from the ontology. This example is taken from the concept *learningObject* that has the attribute *contributeLifeCycle*, which exists in the sLOM-ontology used in LUISA. This attribute points to an instance of the concept *contribute* from the same ontology. This concept in turn has three attributes called *role*, *entity* and *date*. The corresponding part of the ontology is given in listing 4 below:

```
concept learningObject
  contributeLifeCycle impliesType contribute

concept contribute
  role impliesType (1 1) lomRolesVocabularyItem
  entity impliesType (1 *) vCard
  date impliesType (0 1) lomDateTime
```

Listing 4: The concepts ‘learningObject’ and ‘contribute’ with their respective attributes

In this structure we know that we want to edit values of the attribute *role* and *date*. In the case of attribute *entity* we could choose between editing the value directly or the attributes of the concept *vCard*, this will be described below. If a machine were to create an editor out of this information (i.e. without any built-in knowledge about certain attributes or concepts) one of these two simple assumptions would be possible to use:

1. All attributes that are to be edited are defined directly on the concept, which in this case would mean editing the value of the attribute *contributeLifeCycle*,

2. The editing only takes place in the leaves of the hierarchy, i.e. we must descend as far as possible into the metadata structure to find the place to edit.

The second algorithm would not prove useful in the case presented in listing 4, since we know that the instances of *lomRolesVocabularyItem* form a fixed set of instances that should not be altered. Using assumption 1 would also fall short, since the attribute *contributeLifeCycle* would be annotated directly, which would mean that instances of the class *contribute* would be possible to choose from in the user interface. No such instances exist in the ontology, but can of course be created by combing all instances of *vCards* together with all the instances of *lomRolesVocabularyItem*. This would not be very practical and if, for instance, we were to add the attribute *date* to the instance this approach would be extremely unpractical. Here a little bit of additional knowledge is needed in order to construct a useful editor. However, to add this knowledge into the ontology would be inappropriate, since this would add information on a specific use of the ontology, which defeats the purpose of ontologies to provide general descriptions of a domain. This information should instead be provided when using the ontology in a specific way, and Annotation Profiles contain this extra, context-specific information. As mentioned above the attribute *entity* it is not even simple for a human to decide where in the structure to edit. The Annotation Profile could be created to:

- Edit the value of the attribute *entity*, or
- Edit the attributes *email*, *name* and *phoneNumber* of the *vCard* concept (given in the listing 5 below).

```
concept vCard
  email impliesType _string
  name impliesType (1 *) _string
  phoneNumber impliesType (0 2) _string
```

Listing 5: The concept vCard

If this structure needs to be annotated in both ways it is possible to create two separate Annotation Profiles, which would not be possible to do if the editor was created directly from the ontology. As we have seen from this example, it cannot be inferred from the ontology what the user is interested in editing and especially what to include or exclude from the ontology. Moreover, it is sometimes not even possible to make one Annotation Profile that fits every need for an attribute. Here the Annotation Profile construct proves its usefulness, since it acts as a filter for what should be possible to edit and what should not - and more than one such filters can be created for changing different parts of the structure.

We have shown that the information provided in the Annotation Profiles cannot be derived from the ontology. So, the Annotation Profiles have the task to act as input to the editing tool. The APs could be made manually, but that would be rather cumbersome in the semantically rich metadata context of LUISA, so a

tool named *Formulator* has been developed for that purpose. This tool will be described below.

2.3 Using Annotation Profile Snippets – Formlets

As discussed in the section above, not everything in an ontology needs to be annotated. When using Annotation Profiles not all concepts and attributes of a concept need to be included. This means that a number of different Annotation Profiles, where different sets of attributes are annotated, can be applied to the same instance.

When doing this, several attributes might be repeated in every profile. This is the reason behind a construct called a *Formlet*, which has been used to handle this situation. A Formlet is a small Annotation Profile that can be combined with other Formlets to form an Annotation Profile. There are no real restrictions on how big an Annotation Profile or a Formlet can be, but a good rule to follow is to only have one attribute on the starting concept. The Formlets can then be combined into an Annotation Profile. This kind of construction is called a *compound of Formlets*, and it is basically a list that points out the Formlets to be used. During runtime, the formlets of the compound are aggregated into one Annotation Profile. This process is transparent to the editor of the metadata and its purpose is to simplify the role of the Annotation Profile author. An example of Formlets and compounds of them is given in the following subsection.

2.4 Using the Formulator tool

In order to construct the Annotation Profiles, a tool called Formulator has been developed. It is a first prototype of an application where Annotation Profiles can be managed. When starting the Formulator tool, a window that looks something like the screenshot in Figure 1 will appear.

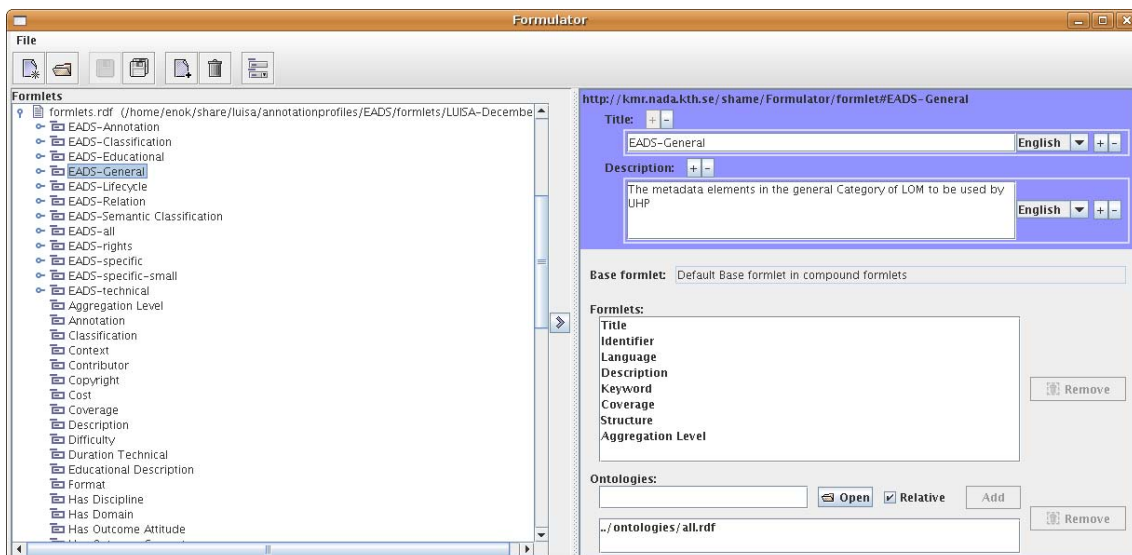
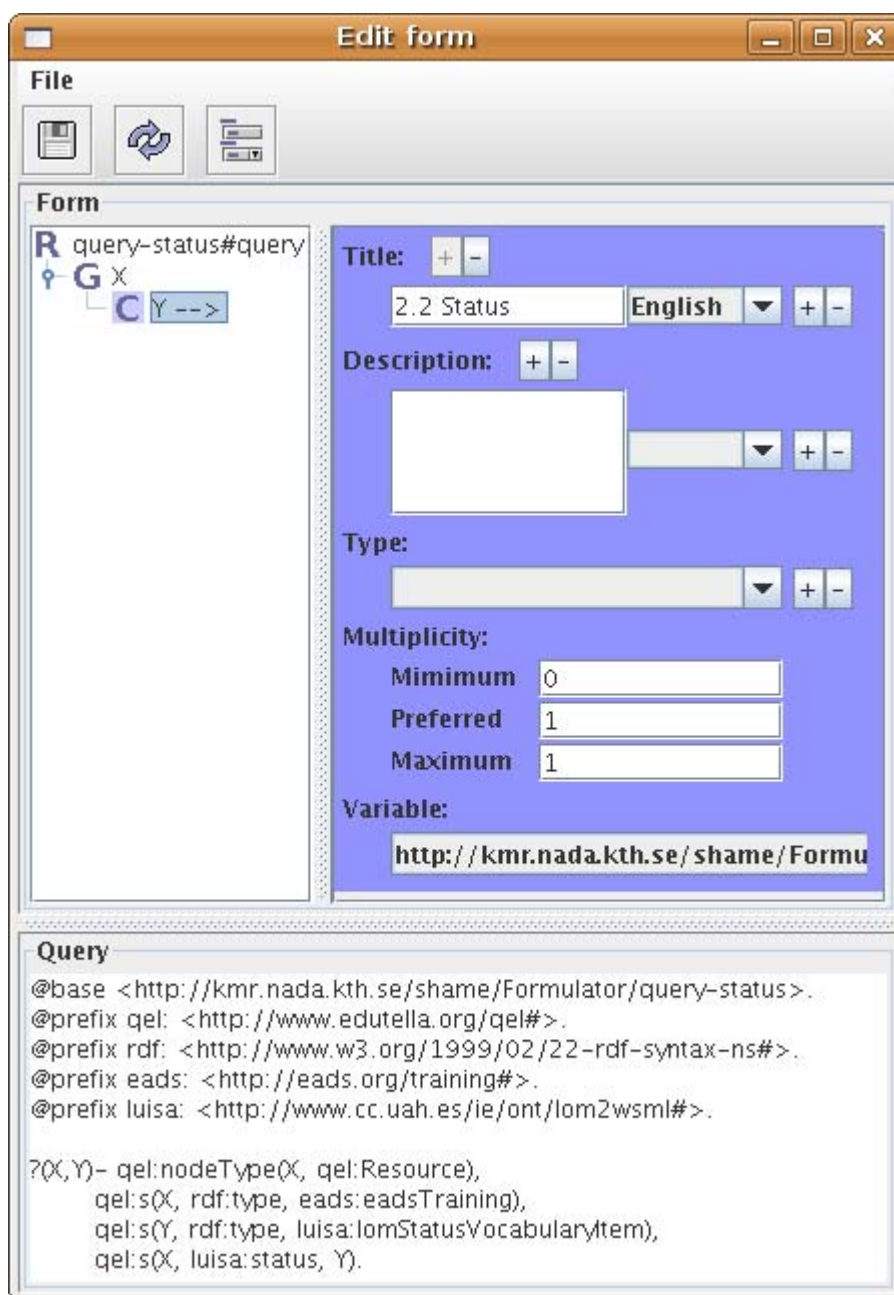


Figure 1: Screenshot of the main window of Formulator

To the left all the Annotation Profiles are listed. The compounds appear at the top with a small blue icon before each one. As described above, a compound can consist of a combination of other compounds and Formlets. In the screenshot above, an editable compound (EADS-General) has been selected from the list and is therefore shown in the top right part of the screen. This compound has been created for the attributes contained in the General Category of LOM.

If we select a Formlet in the list, the right part of the window is changed and it becomes possible to edit the Formlet. The Graph Pattern and the Form Template can then be edited inside a window that pops up (See Figure 2). In the screenshot the Formlet for the attribute status is edited.



Form

R query-status#query
 G X
 C Y-->

Title: + -
 2.2 Status English + -

Description: + -
 [Empty text box] + -

Type: + -
 [Empty dropdown] + -

Multiplicity:

Minimum	0
Preferred	1
Maximum	1

Variable:
 http://kmr.nada.kth.se/shame/Formu

Query

```
@base <http://kmr.nada.kth.se/shame/Formulator/query-status>.
@prefix qel: <http://www.edutella.org/qel#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix eads: <http://eads.org/training#>.
@prefix luisa: <http://www.cc.uah.es/ie/ont/lom2wsml#>.

?X,Y- qel:nodeType(X, qel:Resource),
      qel:s(X, rdf:type, eads:eadsTraining),
      qel:s(Y, rdf:type, luisa:lomStatusVocabularyItem),
      qel:s(X, luisa:status, Y).
```

Figure 2: Editing a Formlet in Formulator

In the screenshot in Figure 2 one can see that the Graph Pattern of the Formlet is edited via a text-field at the bottom of the window. Above that field the Form Template can be manipulated via a GUI. In the left part of the window the structure of the template can be manipulated, and in the screenshot we see a GroupFormItem (marked with 'G') that is bound to the variable X from the Graph Pattern. This GroupFormItem contains a ChoiceFormItem (marked with a 'C') that is bound to the variable Y from the Graph Pattern. In the right part of the screen, the attribute of each item can be edited, in this case the title (which will be the label in the resulting editor), the description of which type, and also the cardinality-constraints.

Our prototype of the Formulator tool is still rather difficult to use, but it is easier than having to create everything manually using the file system as your only support. In the next section, the ideas for how this tool should look is outlined together with other kinds of functionalities that could support the creator of Annotation Profiles.

3 IDEAS FOR IMPROVING THE MANAGEMENT OF ANNOTATION PROFILES

3.1 A Simplified Annotation Profile format

The current design of Annotation Profiles with a separation into a Graph Pattern and a Form Template is somewhat awkward to work with. There are several reasons why this design was chosen – such as:

- It directly corresponds to the internal representation in the API, which ensures that the functionality of the API is exposed.
- Existing query languages can be used for the Graph Pattern.
- It resembles the traditional separation between the model and the view in a Model-View-Controller pattern.
- It allows RDF to be exchanged for another language - as long as the principles of the hierarchical matching and template-based creation process can be supported.

However, these reasons are on the theoretical level and are outweighed by practical experience, where this separation mostly causes confusion and hinders adoption.

A simpler unified format need not affect the internals of the API. Instead it can be treated as an alternative syntax that is translated into a Graph Pattern and Form Template when used. The simplified syntax would be similar to the Form Template, where the variable is replaced with a leading property for each form item. The following example show what such a syntax could look like in XML.

```
<Group id="test:test">
  <Label lang="sv">Resurs</Label>
  <Text property="dc:title" style="languageBoxVisible;">
    <Label lang="en">Title</Label>
  </Text>
  <Choice property="dc:subject" constraint="ns2:Subjects">
    <Label lang="en">Subject</Label>
  </Choice>
  <Ref refId="test:date"/>
</Group>
<Text id="test:date" property="dc:date" constraint="xsd:Date"
      nodeType="DatatypedLiteral">
  <label lang="en">Date</label>
  <label lang="sv">Datum</label>
</Text>
```

Besides being unified, there are several other positive aspects of this approach:

- Variables can be omitted.
- A more direct relationship is established with the underlying data model as all parts needs to be exposed in the syntax. If some things should be hidden from the end users, they are marked as hidden, not left out.
- Breaking up large Annotation Profiles into reusable fragments (Formlets) is simplified. Furthermore, to reuse a fragment via a reference is done in a straightforward manner directly inside of another Annotation Profile (currently special compound Formlets have to be used).

In addition, a single unified format is easier to build good editing tools for. This is true both from a user interface design perspective as well as from the administrative aspects of managing Graph Patterns and Form Templates in pairs.

To simplify even further, a range of common patterns in Annotation Profiles could be identified and provided as possible - and well documented - starting points.

3.2 Improved Annotation Profile management

One of the main arguments for using Annotation Profiles is that you can reuse them. Hence, it is important that an Annotation Profile is easily accessible and

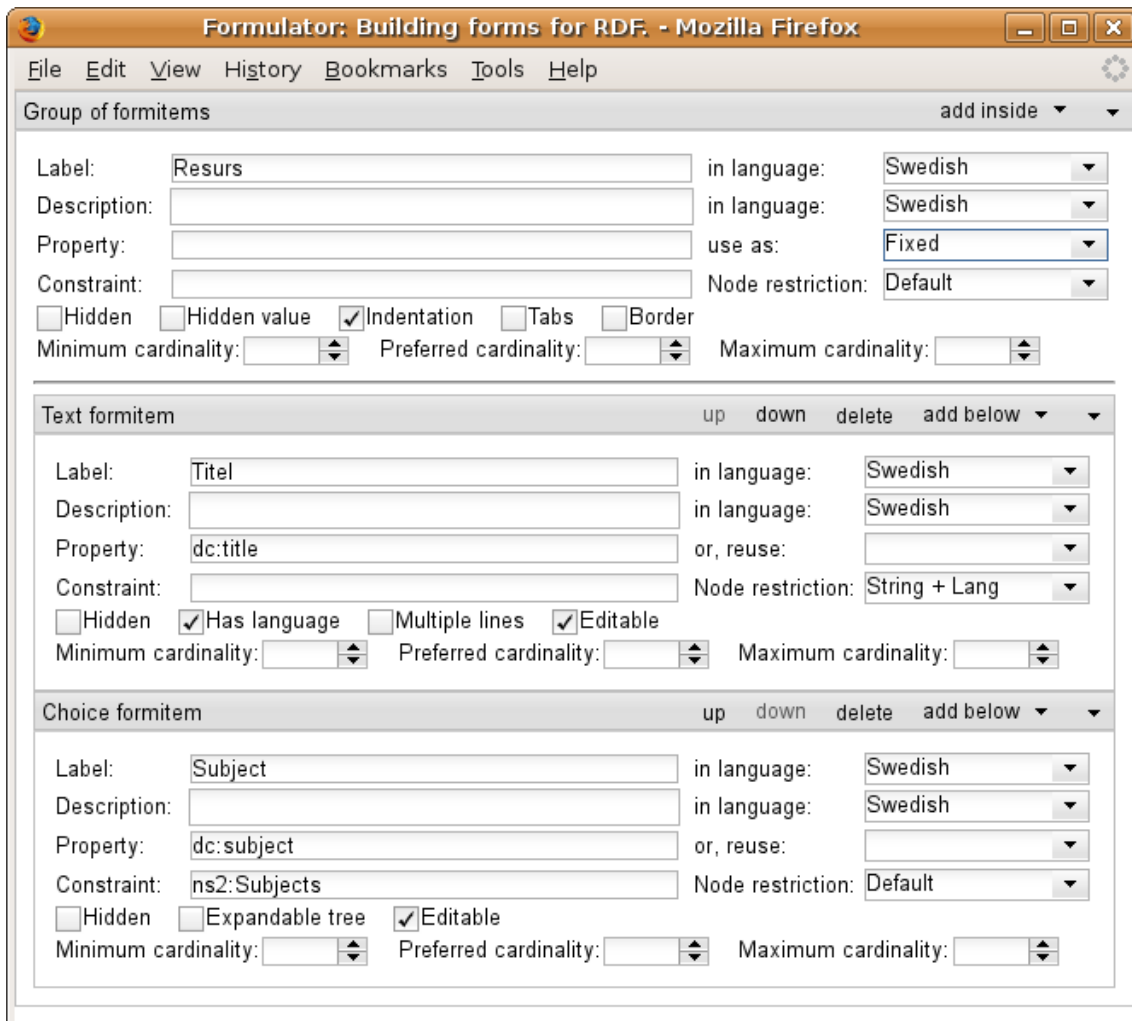
shared in a public environment. Today, Annotation Profile Authors are encouraged to either reuse a few well-known APs that are stored in the project environment alongside with the code, or to create new ones locally. And in the next step, when a metadata editor is deployed, a set of Annotation Profiles are selected, packaged, and then incorporated statically into the tool. This approach is clearly not flexible enough.

Hence, we propose a web-based tool, which could be called the *Web Formulator*, where existing Annotation Profiles can be built, discovered, and then reused as a whole or as fragments. The simplified unified format introduced above will most probably be used, since it would significantly reduce the complexity of the authoring part of the Web Formulator.

This tool could be built with AJAX technology and adhere to the REST paradigm [3]. One implication of this is that all Annotation Profiles will have unique URLs from where they can be retrieved in various syntaxes. This makes it possible to build metadata editors that have no pre-packaged list of Annotation Profiles. Instead such editors can be configured dynamically by referencing Annotation Profiles in the Web Formulator.

This flexibility also imposes new requirements on user management and access control to ensure that established Annotation Profiles are truly stable when so intended.

Below is a first draft of how the Web Formulator's user interface for editing a single Annotation Profile according to the simplified unified syntax introduced above could look like.



The screenshot shows the 'Formulator' web application interface. It is titled 'Formulator: Building forms for RDF. - Mozilla Firefox'. The interface is organized into three main sections, each with a set of controls and a 'Group of formitems' dropdown menu.

- Group of formitems:**
 - Label: Resurs
 - Description: (empty)
 - Property: (empty)
 - Constraint: (empty)
 - in language: Swedish
 - in language: Swedish
 - use as: Fixed
 - Node restriction: Default
 - Options: Hidden, Hidden value, Indentation, Tabs, Border
 - Cardinality: Minimum, Preferred, Maximum (all empty)
- Text formitem:**
 - Label: Titel
 - Description: (empty)
 - Property: dc:title
 - Constraint: (empty)
 - in language: Swedish
 - in language: Swedish
 - or, reuse: (empty)
 - Node restriction: String + Lang
 - Options: Hidden, Has language, Multiple lines, Editable
 - Cardinality: Minimum, Preferred, Maximum (all empty)
- Choice formitem:**
 - Label: Subject
 - Description: (empty)
 - Property: dc:subject
 - Constraint: ns2:Subjects
 - in language: Swedish
 - in language: Swedish
 - or, reuse: (empty)
 - Node restriction: Default
 - Options: Hidden, Expandable tree, Editable
 - Cardinality: Minimum, Preferred, Maximum (all empty)

Figure 3: First draft of Web Formulator authoring part.

3.3 Creating skeleton Annotation Profiles from ontologies

If you need to create an Annotation Profile from scratch, the simplified syntax and the better management of Annotation Profiles via the Web Formulator tool would certainly be useful. However, for those that have gone through the process of creating a formal ontology there is much to gain if the information in such an ontology could be used in the creation of new Annotation Profiles. The alternative, to place this responsibility with the Annotation Profile author, is not only extra work, but it is also error-prone, since copying this information manually is a tedious task. Hence, we suggest to support automatic creation of skeleton Annotation Profiles from ontologies.

To arrive at useful Annotation Profiles there are two basic problems to overcome. First, how is the scope of each Annotation Profile decided (i.e. how deep should it go)? Second, how should the skeleton Annotation Profile be divided into fragments that make modifications simple?

We propose to select a list of top-level classes from the ontology that will act as delimiters to the creation process. From this list the creation process will result in one Annotation Profile per top-level class, and for each top-level class there will be one Annotation Profile fragment per outgoing property. The Annotation Profiles for the top-level classes will reference all corresponding fragments in an arbitrary order. A typical modification after this creation process is to change the order of the Annotation Profile fragments and maybe leave some of them out. The scopes of the Annotation Profiles are decided by the referenced fragments. In general each fragment may be arbitrarily deep by traversing multiple ontology classes. It will not be allowed to traverse into the properties of one of the other top-level classes.

3.4 Validating Annotation Profiles against ontologies

Independently of how an Annotation Profile has been created, it is valuable to be able to verify that it will create valid instances of a specific ontology. This is especially useful when ontologies are developed independently - or just by someone else than the Annotation Profile author.

The details of this validation process can be largely inferred from deliverable 3.2 (The Annotation Profile Specification) [1], where it is stated which information from ontologies that should be interpreted in the context of an Annotation Profile. For example, if there are properties that are enforced on a specific class in the ontology, they must be included (with appropriate cardinality restrictions) in the Annotation Profile at all the places where this class is given as constraint.

4 EXPERIENCES FROM EDITING (AND IDEAS FOR IMPROVING)

One of the main benefits of using Annotation Profiles is that the responsibility for which metadata structure that can be edited is shifted from the code developer to the Annotation Profile Author. For the end-user, the use of Annotation Profiles should be transparent. Some of the drawbacks that we have experienced in LUISA are that some special requests from the use-case partners on how certain attributes should be displayed or edited could not always be fulfilled. This is because some of the information required to fulfill the request does not exist in the Annotation Profile. In this section we will describe the problems that we have faced and also outline an approach to solve them.

4.1 Complexity of deep data-structures

When designing the user-interface of an Annotation Tool it is a good idea to try to hide as much as possible of the complexity of the underlying structures. However, there is a trade-off between making everything editable according to the ontology on one hand, and displaying a simple layout on the other. Here is an example to illustrate this problem, where the attribute *descriptionLO* on the Learning Object is edited. The corresponding editor is shown in Figure 4 below.

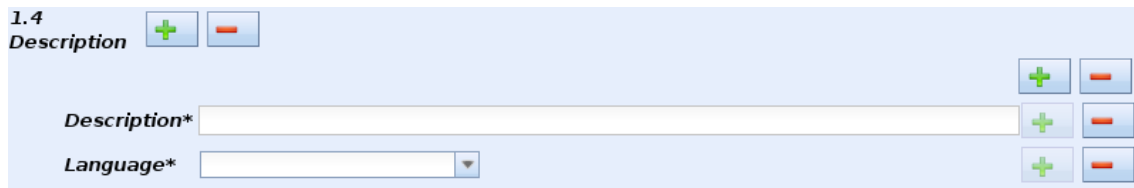


Figure 4. An editor where the description of an LO is edited

As can be seen in this screen-shot, we have 4 pairs of '+' and '-' signs for just editing a description. Can this complexity be removed somehow? If we take a look at the structure in WSMML for the attribute description (listing 6), we see that the structure in the ontology is basically reflected in the Annotation Tool. However, this is still confusing for an end-user, who cannot be expected to have full knowledge about the structure of the ontology.

```

concept learningObject
  descriptionLO impliesType (0 *) langString

concept langString
  hasSingleLangString impliesType (0 *) langStringSingle

concept langStringSingle
  hasHumanLanguage impliesType (1 *) ocHumanLanguage
  hasCharacterString impliesType (1 *) _string
  
```

Listing 6: The structure used for the attribute descriptionLO

In this structure the values of the attributes *hasHumanLanguage* and *hasCharacterString* are to be edited, starting from the concept *learningObject*. This involves an intermediate concept called *langString*. Looking at this structure one can understand why the interface looks like it does. As the intermediate concept *langString* is not visible in the Annotation Tool, the signs for adjusting the cardinality could be removed here which could be easily arranged in the Annotation Profile. But, this would also mean that the editing tool would become more restricted than needed. However, it is a good idea to not display any signs if the minimum cardinality is set to 0 and maximum set to 1.

4.2 Use of controlled vocabularies

The values that are not editable by using a text-field are created by choosing a value from a finite set of choices. This is usually achieved by using a drop-down menu in the user interface which works well in the case where the set of choices is explicitly expressed inside the Annotation Profile. In the case where the set of choices is indirectly given, i.e. as a constraint in the Graph Pattern, the list can be rather long if many resources match the constraint. If the type of the value has some kind of hierarchical structure, the values can be sorted

according to this hierarchy. This was implemented by using a tree that showed the hierarchy from which the value can be chosen, which was requested by the use-case partner EADS. Because their set of possible choices for some attributes became too large, it was not practical to choose one value from a drop-down menu.

There is also the opposite situation, when the set of choices are limited to very few options. In such situations it was requested that radio-buttons should be used to select the value. So, radio-buttons are rendered when the number of choices is four or less (this number is configurable), and the maximum cardinality is set to one. With a higher maximum cardinality it would be quite natural to use check-boxes, but this is not supported yet.

In the case where the number of choices are too large to use a radio-button or the set of choices do not have any hierarchical structure, the way to choose the value defaults to a drop-down menu. To make the annotation process easier it would be nice to sort the values, for example with the most frequently used values at the top. No such information can be expressed in the Annotation Profiles, but some of these arrangements can be done in the code of the Annotation Tool, like ordering the labels of the values in alphabetical order.

4.3 Need for check of constraints and for required values

In some systems it is a requirement that some values are unique. An example of this was presented by the use-case partner UHP, which wanted to have a check that the same title was not used by more than one LO. This would require a communication between the saved Learning Objects and the Annotation Tool. This is not really possible to implement in the present design, since no such information exists in the Annotation Profile. If every value is required to be unique it is of course possible to implement this requirement directly in the code, but such a requirement is rare.

The use-case partner EADS had a similar request, where some attributes were compulsory and if they were not filled in, the edited metadata should not be possible to save. Information that does exist in the Annotation Profile could be used to fulfill this request, but the solution would have to be implemented in code. Say that the minimum cardinality of an attribute is one or more, then it could be assumed that this attribute is required and if a value is given the Annotation Tool would not accept a save-request. Even if the minimum cardinality is set to zero in the ontology, a higher value could be set in the Annotation Profiles. In this case the AP sets a higher constraint than the ontology and this is allowed, but of course not the other way around. This way the requirement could be fulfilled by changing the APs used. However, this solution causes a problem in the case where the value of an attribute is not considered compulsory, but has a minimum cardinality of one in the ontology. From our experience this is not a common case, but could of course still occur. One possible solution is to automatically create these values or have a default value. These aspects are considered further in the coming subsections.

4.4 Need for default values

There are some cases where the same value of an attribute will occur frequently, for example attribute *hasHumanLanguage* given in listing 6 given above. This attribute tells the language of the value of the attribute *hasCharacterString* for the same instance. If only one language is used every time a lot of time can be saved by setting a default value. Unfortunately this is a bit difficult to implement, since the required information cannot be retrieved and is not a part of the Annotation Profile. This case could be partly implemented in the code, for example, the requirement that every string that has a language is set to be in a specific language.

4.4.1 Automatic creation of values

Closely related to default values is automatic creation of metadata. There are some attributes where the value can be automatically generated. For example, there is a field in the metametadata category in LOM that has an attribute called *creationDate*. This is an example where the value of the attribute can be found rather easily. This is an interesting area of research some interesting mechanisms to automate the values of attributes exists, but such mechanisms have not been considered further in LUISA, since our focus has been on creating a tool that supports human editing of the metadata.

4.5 Connecting Learning Objects

As described in section 4.2 the value of an attribute that is calculated from the ontology can be chosen in the user interface. The ontology to query is referred to from the Annotation Profile, but this part has not yet been fully adapted to the web context as the rest of the Annotation Tool. Currently the ontologies need to be placed on the same machine as the Annotation Tool and packaged together with the Annotation Profiles.

This means that the only instances that can be found when querying the ontology are the one that are fixed. So, in the case where an attribute has a value that is from a dynamic set of instances (for example when a Learning Object refers to another Learning Object) is presently not supported by the Annotation Profiles. Such support would require a live connection to a repository, where those instances could be searched for. It also means that new versions of the ontology (for example when the fixed set of instances are updated) requires that the ontology used by the Annotation Tool needs to be updated. This will be solved if a web-solution could be worked out, but that would require a careful solution to not cause problems of long waiting times due to many queries sent to server holding the ontology.

4.6 To edit many resources in a repository

Editing many resources in a repository is actually not directly related to the use of Annotation Profiles, but rather to the management of instances. In Figure 5 the ELUISA interface, which was the first prototype of the Annotation Tool, is displayed. A number of LOs are listed, and if one of them is clicked, an editor for that resource is displayed. This is a very basic interface, which is sufficient

for handling a small number of resources. However, it becomes inconvenient when we have more than 20 resources in this list. A kind of sorting mechanism was therefore requested - for example to sort by date of creation, by language or by any other field. These requests have been partly implemented in the second version of the tool.

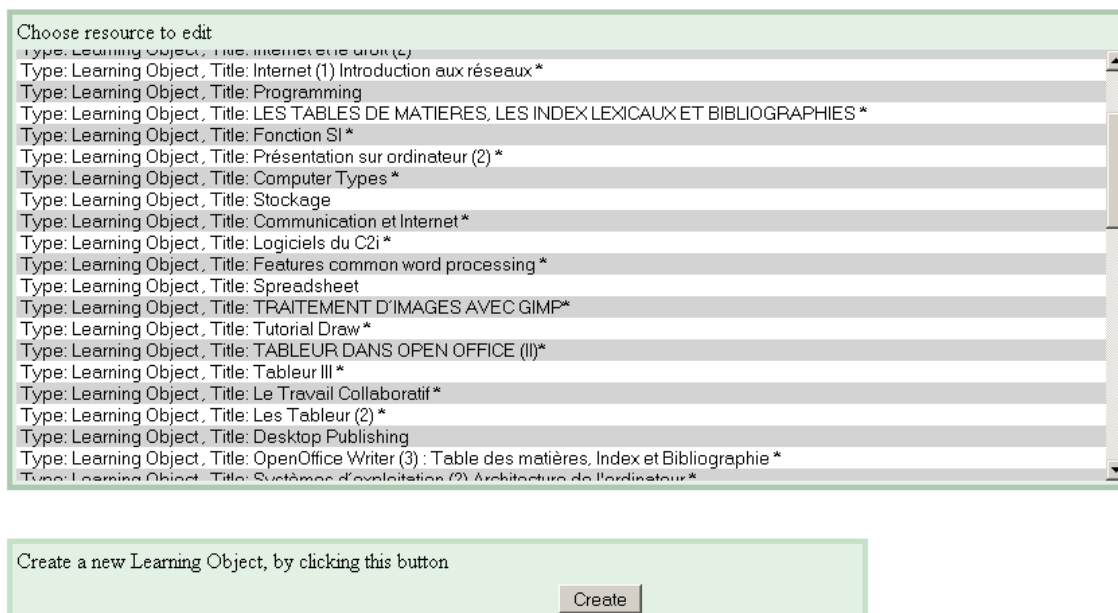


Figure 5: Managing the metadata of LOs

5 IDEAS FOR IMPROVING THE EDITING EXPERIENCE

In the previous section some ideas on how the editing experience can be improved have been presented. Some of these ideas have been implemented in the Annotation Tool, when the required information could be derived or calculated from the Annotation Profiles or the underlying ontologies sometimes with a few (minor) extra assumptions. But these requests are only from the two use case partners in this project. Other users will probably have other requests on the functionality.

5.1 Extending the Annotation Profile

As discussed earlier, the main reason that certain functionalities could not be implemented was that the Annotation Profile does not contain the required information. As we have seen in such cases, one option would be to implement the requested suggestion in the code, which we have done for example in the case when a hierarchical structure could be identified from the ontology. In the case of having a default-value for a specific attribute, a solution in code would require a lot of specific knowledge about the metadata structure and then the idea of hiding the specifics of the metadata structure from the developer would be lost. To include such information in the Annotation Profile would be a possible solution, but that way the APs would become too specific to be reused by different users or annotation tools. In the example given in section 4.3 the language would have set by default, say to example French. An AP with that

default value would not be considered directly reusable in an application where the default value should be Swedish.

A solution that we will continue to investigate is to extend the description by allowing descriptions that are located outside the Annotation Profile. As stated in LUISA deliverable 3.2 [1] the Annotation Profile Model is not a complete description on how to build an editor. We will continue to define more exactly what should be included in the Annotation Profile Model and how the extended descriptions could be added.

The idea is that the extensions are given in the Annotation Profile, but it is not a requirement for an application to support them. So in case the application does not recognize the extensions they are simply ignored. In this case it is also necessary to consider how Annotation Profiles will be used. To aim for reusability is of course a good thing from a design point of view, but is it something that will be used? This question is hard to answer without a more widespread adoption of Annotation Profiles.

6 CONCLUSION

The first conclusion that can be drawn from using Annotation Profiles in LUISA are that this mechanism makes the creation of several different editors easier, as only one tool had to be created that follows the AP specification. With the input of different APs different editors were then automatically created. It also takes away the responsibility of standards compliance from the developer of the tool and this can now instead be handled by the role that we choose to call *Annotation Profile author*. Now to support this role the tool we call *Formulator* has been developed, but this tool is currently not so user friendly and requires a lot of specific knowledge. Therefore a Web-formulator is under development that will also set the creation of Annotation Profiles in a web oriented context. This would include a mechanism that created an "AP skeleton" that can derive the information that exist in the ontology. Together with that a confirmation mechanism that calculates if an AP will can be used to edit a structure that exist in the ontology

We also see it necessary to conclude that it is in some cases harder to tailor specific ways of editing, for example having a default value for a specific attribute. That was one of the request on the Annotation Tool from the use-cases in the project that could not yet be fulfilled. Simply because the information necessary could not be found or derived from the Annotation Profiles or ontologies. Some of the requests could be taken care of in the code, but those are typically (and should be) requests that are general and does not apply to a specific attribute or value.

So the continued work with Annotation Profiles will continue and we will look into how specific needs can be handled, as there are probably a lot more requests from other potential users that we can not include in the Annotation Profile, as it would make the Annotation Profile model too complex. One solution was presented in the document where extensions could be added, and in the case that they cannot be recognized they will simply be ignored. Together with that a solution to have access to ontologies over the web to populate the

drop-down menus, radio-buttons and hierarchy trees when the value needs to be selected from an instance in an ontology.

REFERENCES

[1] Palmér, 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] Palmér, M.; Enoksson, F.; Nilsson, M.; Naeve, A. (2007) *Annotation Profiles: Configuring forms to edit RDF*, Proceedings of the international conference on Dublin Core and metadata applications, Singapore, ISSN:1939-1358

[3] 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>