University of Innsbruck

Semantic Technology Institute

An Annotation System based on the SA Methodology

Bachelor Thesis

Christian Ammendola

Matriculation number: 0316635

Supervised by Dr. Elena Simperl

Co-supervised by Tobias Bürger

July, 2008

Abstract

This bachelor thesis is based on the SA Methodology, a manual annotation approach based on ontologies as presented in [1]. The methodology consists of multiple parts which support the user in the selection of ontology elements and in the extension of ontologies, if elements needed for annotation are missing. This document presents a prototypical implementation of an annotation system providing capabilities for ontology-based metadata creation and a partly implementation of the SA Methodology.

Contents

1 Introduction

The creation of ontology[1]-based annotations is generally acknowledged as being an important task in order to get a step closer to Tim Berners-Lee's vision of the Semantic Web [2]. For the creation of annotations manual, semi-automatic, and automatic approaches [9] exist. On the one hand, automated strategies represent a fast alternative for the creation of annotations, but they mostly don't deliver satisfying results. On the other hand, most manual approaches are not adequate for users having no experience in dealing with ontologies. We present a solution for supporting users in this process in [1] and provide a manual annotation approach which is suitable for end users: the Element **S**election and Element **A**ddition (SA) Methodology.

The methodology is divided into two parts as depicted in figure 1: the *Element Selection* and the *Element Addition* part. The first part deals with the support for users in the selection of adequate ontology elements. The execution of this task can be difficult when the ontologies include large hierarchies of elements, especially for users who are not familiar with them. The second part faces the problem of extending ontologies which often do not include all elements which are needed for annotation. Therefore, the methodology provides an approach for extending ontologies during annotation time without the need of being a domain or ontology expert.

Each part of the SA Methodology includes different components in order to achieve its objectives. The *Element Selection* part consists of the *Ontology Browser*, the *Ontology Filter*, the *Semantic Search with Autocompletion*, and the *Recommendation System* components and the *Element Addition* part of the *Ontology Extension* and the *Classification Support* components. Not all of them are mandatory in order to achieve the goals of the methodology, however, the combination of some of them is recommended.

This thesis, based on the aforementioned approach for the creation of metadata, realizes an annotation system consisting of an annotation frame-

[1]A detailed presentation of ontologies is provided by [8].

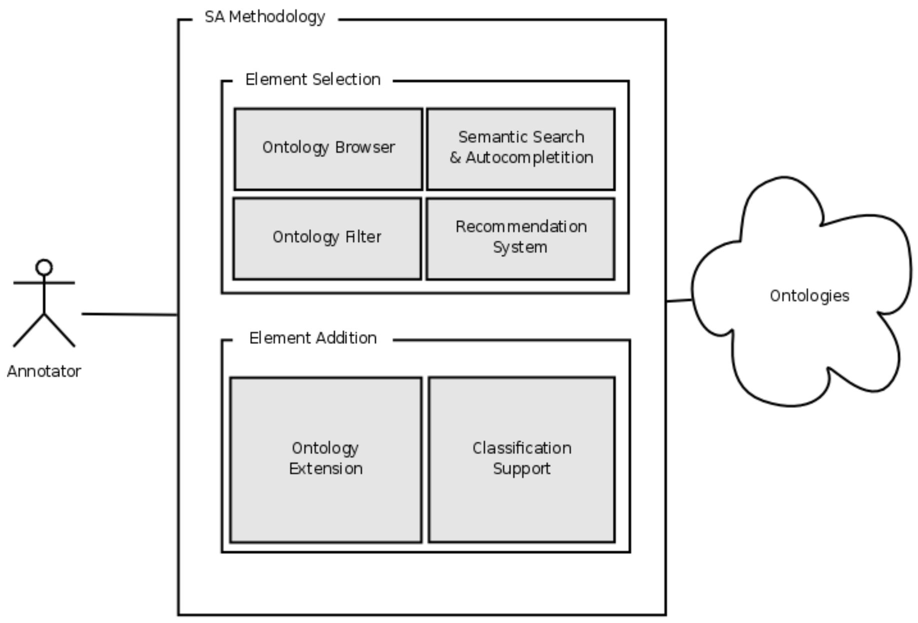


Figure 1: SA Methodology Overview

work and an annotation client, both of which realize some components of the methodology.

The objectives of the thesis are listed below:

- To provide a prototypical implementation of an ontology-based annotation framework including capabilities for element selection support and work-embedded element addition.
- To implement an annotation client for end users based on the annotation framework.

For the realization of the element selection support we use the *Semantic Search with Autocompletion* component and for the element addition we implement the *Ontology Extension* and the *Classification Support* components which offer the possibility to add new concepts and relations. The classification of newly added elements is realized by the manual classification approach of the SA Methodology.

The remainder of this document is structured as follows. Section 2 presents the annotation system including its requirements and technical realization. Finally, concluding considerations are discussed in section 3.

2 Annotation System

In this section we present the annotation system, including the annotation framework and the annotation client, in more detail. Figure 2 shows the main architecture of the annotation system. It includes:

(1) The annotation framework consisting of a set of Web services[2] offering capabilities for ontology-based annotation including functionalities enabling a partly realization of the SA Methodology.

(2) The annotation client providing an environment that can be used by end users for manually creating annotations. The client is based on the capabilities of the annotation framework.

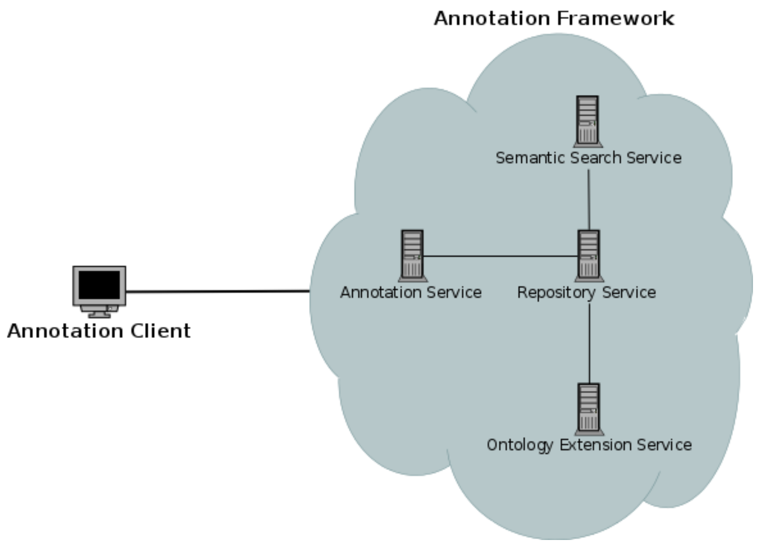


Figure 2: Annotation System Architecture

[2]http://www.w3.org/2002/ws/

In the next sections the different parts of the system are going to be presented in more detail: In section 2.1 we present the requirements posed to the annotation system, in section 2.2 we discuss its technical realization, in section 2.3 the Web services constituting the annotation framework are presented and in section 2.4 the annotation client is illustrated.

2.1 Requirements

In this section we present the requirements of the annotation framework and the annotation client. Both applications should enable the manual annotation and the support of ontology-based annotations about resources. Aid in the annotation process is realized based on *Semantic Search with Autocompletion*, the *Ontology Extension*, and the *Classification Support* components of the SA Methodology. In the following the list of requirements posed to the annotation system is presented:

- Selection of local resources[3] to be annotated.
- Creation of annotations about selected resources.
- Persistent storage of annotations.
- Management (loading, editing) of annotations about a resource.
- Selection of concepts and relations from ontologies used for the annotation, supported by a semantic search with autocompletion capability.
- Extension of the vocabulary of the ontologies by the addition of new concepts or relations.

2.2 Technical Realization

The annotation framework and the annotation client are realized using technologies as presented below:

[3]In this thesis with local resources we mean files stored on a local harddisk.

- *Java:* We used the Java programming language[4] for the implementation of both the annotation framework and the annotation client. A more detailed presentation about the Java programming language is provided in [11].

- *Axis2:* Axis2 is a framework for the implementation of Web services, based on the Java programming language. We used Axis2 for the implementation of the annotation framework. A detailed presentation of Axis2 can be found in [7].

- *Eclipse RCP:* For the implementation of the annotation client we used Eclipse RCP, a platform for building desktop applications in Java. A more precise presentation about Eclipse RCP is provided in [13].

- *WSMO:* The Web Service Modeling Ontology (WSMO)[5] is an ontology providing capabilities for describing Semantic Web services. We designed the annotation framework on the basis of on the WSMO model. Further details about WSMO are provided in section 2.2.1.

- *WSML:* The Web Service Modeling Language (WSML)[6] is a language that among others provides capabilities for modeling ontologies and is based on the WSMO model. We used WSML for the implementation of ontologies necessary for representing annotations. A more detailed presentation about WSML is provided in section 2.2.2

- *WSMO4J:* For the usage of WSML constructs within a Java environment we used WSMO4J[7], a Java API providing access to the elements implemented in WSML. A more detailed presentation of WSMO4J is provided in section 2.2.3.

[4]http://java.sun.com/

[5]http://www.wsmo.org/

[6]http://www.wsmo.org/wsml/

[7]http://wsmo4j.sourceforge.net/

- *ORDI:* The Ontology Representation and Data Integration (ORDI) Framework is a middleware enabling the persistent storage of WSMO elements. A more detailed presentation about ORDI is provided in section 2.2.4.

In the subsequent sections we introduce the mentioned technologies in more detail.

2.2.1 WSMO

The Web Service Modeling Ontology (WSMO) is a model for Semantic Web services (SWS), which essentially are a combination of Semantic Web technologies and Web services [6]. The aim of SWS is to enable an automated discovery, composition, and execution of Web services amongst others based on semantic descriptions of various aspects of Web services.

The WSMO model [6] includes the following elements for these descriptions:

- *Ontologies:* Ontologies can be used in order to create vocabularies with formal semantics. The terminology, which is provided by ontologies, can by used for descriptions by the other WSMO elements.
- *Web Services:* The WSMO web service element can be used to describe the functionalities provided by a Web service semantically in order to enable its usage in an automated way.
- *Goals:* Goals can be used for describing the objectives of a Web service consumer.
- *Mediators:* The mediator element provides capabilities for the interoperation among the other elements by resolving mismatches between them.

We will focus on WSMO ontologies which are the only elements needed in the annotation system. Ontologies provide capabilities for the formal definition of terminologies and its semantics. Constructs that can be used for the definition of ontologies in WSMO are:

- *Concepts:* Concepts are used to define the terminology of an ontology. For example, *Person* could be a concept.
- *Instances:* Instances are concrete individuals of concepts. For example, *John Doe* could be an instance of the concept *Person*.
- *Relations:* Relations are used in order to describe connections among concepts or instances. For example, *hasFather* could be a relation for connecting an instance person with its father.
- *Axioms:* Axioms can be used in order to specify constraints among concepts and relations.

2.2.2 WSML

The Web Service Modeling Language (WSML) is a declarative language that can be used to describe Semantic Web services based on the WSMO model [4]. The definition of WSML provides different variants of the language based on different logical formalisms such as description logic, logic programming, and first-order logic. These variants have different expressive power.

For the representation of WSML constructs there are different syntaxes: the human-readable syntax, the XML syntax, and the RDF[8] syntax. While the former is a syntax which can be read easily by humans, the two latter are more suitable for machines and are intended for the exchange of WSMO descriptions among computers. Additionally, there is a specification defining the mapping between WSML and OWL[9] [4, section 11].

For the identification of WSMO elements such as ontologies, web services, goals, mediators, concepts, relations, etc. International Resource Identifiers (IRIs)[10] are used.

[8]Resource Description Framework (RDF): http://www.w3.org/TR/REC-rdf-syntax/
[9]OWL Web Ontology Language: http://www.w3.org/TR/owl-features/
[10]http://www.ietf.org/rfc/rfc3987.txt

2.2.3 WSMO4J

WSMO4J is a Java API and reference implementation for the creation of Semantic Web Service applications that rests upon the WSMO model [5].

The API includes a set of interfaces for the management of WSMO elements. A central interface is *Entity* that represents each WSMO element that can be identified such as ontologies, web services, mediators, goals, concepts, relations, etc. For the identification IRIs are used. A specialization of the *Entity* interface is *TopEntity*. It provides additional methods for the processing of WSMO ontologies, web services, mediators, and goals.

Besides classes for creating, editing, and deleting WSMO elements, the WSMO4J API also provides serializers and parsers which support the human-readable syntax, the XML syntax, and the RDF syntax of WSML.

2.2.4 ORDI

The Ontology Representation and Data Integration (ORDI) framework[11] is a middleware that among others enables the persistent storage of ontologies in different data-sources such as files or databases. The idea of the framework is to provide ontology language neutrality and it is constructed on the basis of its own data model being a generalization of the RDF data model. Therefore, each valid RDF model is also a valid ORDI model. For the support of WSML the RDF representation of the language is used by the framework that can be mapped to the ORDI model. At the moment of writing this thesis, the only formalism that is supported by the persistence framework is WSML, but OWL support is planned to be integrated. A more detailed description of the framework is provided by [14]. As WSMO4J has not been designed for the management of persistent storage of WSMO elements, ORDI is used for that purpose.

We will now provide a more detailed description of the annotation framework

[11]http://www.ontotext.com/ordi/

and its client. In section 2.3 we illustrate the capabilities of the annotation framework by presenting its different services; in section 2.4 we present the client enabling manual annotation for end users based on the annotation framework.

2.3 Annotation Framework

The annotation framework is a software system which can be used to realize annotation tools. It supports some aspects of the SA Methodology thus supporting the selection and addition of ontology elements. As already mentioned before, the annotation framework consists of a set of Web services referred to as annotation services in the following. Figure 3 presents an overview of the different services and the main functionalities they offer:

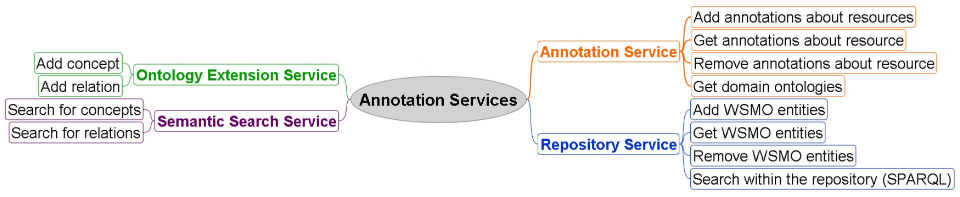


Figure 3: Annotation Services Overview

(1) The *Repository Service* is the core service and as such responsible for the persistent storage of WSMO entities and plays a central role because all the other services rest upon its functionalities.

(2) The *Annotation Service* provides capabilities for the management of annotations.

(3) The *Semantic Search Service* provides functionalities for the search of WSMO entities stored in the repository based on a semantic keyword search.

(4) The *Ontology Extension Service* provides functionalities for the addition of new concepts and relations to the vocabulary provided by the

ontologies stored in the repository.

2.3.1 Repository Service

The *Repository Service* is the core service within the annotation framework which offers an interface for the persistent storage of WSMO ontologies, web services, mediators and goals. It rests upon the Ontology Representation and Data Integration (ORDI) framework introduced in section 2.2.4 and represents the basis of the other services by providing access to the stored elements.

Based on the capabilities of ORDI, the following functionalities are offered by the *Repository Service*:

- File based persistent storage of WSMO elements such as ontologies, web services, mediators and goals.
- Loading of stored WSMO elements from the repository.
- Removal of stored WSMO elements from the repository.
- Execution of SPARQL queries for the retrieval of elements stored in the repository.

A detailed description of the interface of the *Repository Service* is given in appendix A.1. Due to the fact, that only the WSMO ontology elements are relevant for the annotation framework, the methods related to web services, mediators, and goals are not used by the other annotation services.

2.3.2 Annotation Service

The *Annotation Service* offers functionalities for the management of ontology-based annotations. It provides the possibility of adding, loading and removing annotations. The created annotations are stored persistently by the use of the *Repository Service* introduced in section 2.3.1. A detailed description of the interface of the *Annotation Service* is provided in appendix A.2.

The model for structuring annotations is based on the Annotea Annotation Schema[12]. This schema is an RDF based annotation model providing the RDF class *Annotation* used in order to identify annotations. Sub-classes of *Annotation* can be created in order to refine the type of an annotation. For the description of annotations the schema provides a set of properties such as *annotates*, *author*, *body*, *context*, *created*, *modified*, and *body*. Thereby, the property *annotates* is for the specification of the resource is being annotated and the properties *related* and *body* can be used for the description of the content of annotations. A more detailed presentation about the Annotea Annotation Schema and its elements is provided by [12].

In this thesis we used the Annotea annotation model. We created a WSMO ontology with the elements *Annotation*, *annotates*, and *related* of the Annotea Annotation Schema and we added some additional elements. Table 1 presents the elements of this ontology being referred to as the *Annotation Ontology* in the following. The element names listed in the first column of the table, that are written in italic, are the new elements. All of them are specializations of the concept *Annotation* and are used in order to distinguish the type of resource being annotated. Hence, for the annotation of images the *ImageAnnotation* concept should be used. The element *related* is modelled as relation, because there shouldn't be restrictions on the type of the values of the relation. For attributes the specification of the type of allowed values is mandatory.

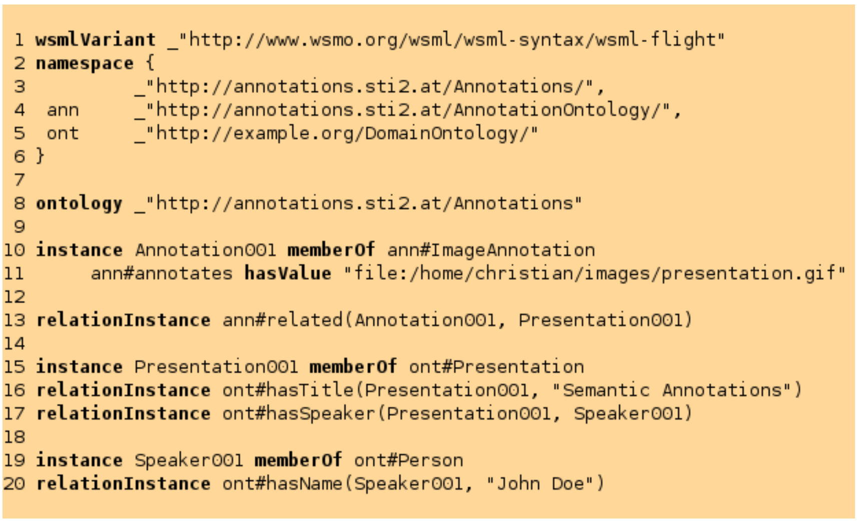
[12]http://www.w3.org/2000/10/annotation-ns

Element name	**Parent element**	**Element type**	**Description**
Annotation		Concept	The type of an annotation resource.
ImageAnnotation	Annotation	Concept	The type of an image annotation resource.
AudioAnnotation	Annotation	Concept	The type of an audio annotation resource.
VideoAnnotation	Annotation	Concept	The type of a video annotation resource.
TextAnnotation	Annotation	Concept	The type of a text annotation resource.
annotates		Attribute	Attribute of the concept Annotation used for specifying the resource being annotated. Each annotation instance has exactly one attribute *annotates*.
related		Relation	Used to describe an annotation. In order to enforce the type of relation between the connected objects, this element can be specialized (e.g. *depicts* could be a specialization saying that the annotated resource depicts something.)

Table 1: Annotation Ontology Elements

The representation of annotations rests upon the use of the annotation ontology. Thereby, each instance of the concept *Annotation* or one of its sub-concepts represents an annotation about exactly one resource that is specified by the help of the attribute[13] *annotates*. Connections between the annotation and other ontology elements can be created by the relation *body*.

Figure 4 depicts an example of an annotation in WSML human-readable syntax based on the annotation ontology of table 2. Its an annotation about a photo depicting a person doing a presentation. At the beginning of the example some general ontology settings are specified such as the WSML variant in line 1 and the identifier of the ontology in line 8. Line 10 defines the annotation with the identifier *Annotation001* being of the type *ImageAnnotation*. With the help of the *annotates* attribute the resource being annotated is specified. In line 13 the annotation is connected to the instance *Presentation001* being of type *Presentation*. This reveals that the annotation tells something about a presentation. Further information about the presentation instance such as the title and the speaker of the presentation is specified in the remaining lines.



```
 1 wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"
 2 namespace {
 3           _"http://annotations.sti2.at/Annotations/",
 4  ann      _"http://annotations.sti2.at/AnnotationOntology/",
 5  ont      _"http://example.org/DomainOntology/"
 6 }
 7
 8 ontology _"http://annotations.sti2.at/Annotations"
 9
10 instance Annotation001 memberOf ann#ImageAnnotation
11      ann#annotates hasValue "file:/home/christian/images/presentation.gif"
12
13 relationInstance ann#related(Annotation001, Presentation001)
14
15 instance Presentation001 memberOf ont#Presentation
16 relationInstance ont#hasTitle(Presentation001, "Semantic Annotations")
17 relationInstance ont#hasSpeaker(Presentation001, Speaker001)
18
19 instance Speaker001 memberOf ont#Person
20 relationInstance ont#hasName(Speaker001, "John Doe")
```

Figure 4: Annotation Example

[13]Concepts in WSMO can have attributes. Attributes are similar to relations but they belong to a specific concept.

2.3.3 Semantic Search (Element Selection) Service

The *Annotation Service* and the *Repository Service*, which are presented in the previous sections represent the base for creating persistently stored ontology-based annotations. The services described in this section and the next section provide functionalities for the realization of the SA Methodology.

The *Semantic Search Service* provides the functionality to retrieve elements such as concepts and relations. The functionality can be used to realize the *Semantic Search with Autocompletion* component of the *Element Selection* part of the SA Methodology.

The service implements a keyword search returning found concepts and relations based on an inserted keyword. Thereby, the keywords are integrated by the *Semantic Search Service* into predefined SPARQL queries that are executed by the *Repository Service*. Figure 5 depicts a SPARQL[14] query for the retrieval of elements with the name *event* being a concept or a relation. The variable *?element* represents the identifier of the found element and the variable *?type* the type of the element, that can be concept or relation. The regular expression *[/|#]event$* is for matching the local name of IRIs that has to be *event*.

```
SELECT ?element ?type
WHERE {
 ?element <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type
 FILTER (
   regex(str(?element), \"[/|#]event$\", \"i\") &&
   (
     sameTerm(?type, <http://www.wsmo.org/wsml/wsml-syntax#Concept>) ||
     sameTerm(?type, <http://www.wsmo.org/wsml/wsml-syntax#Relation>)
   )
 )
}
```

Figure 5: SPARQL Example

The overall process of retrieval of ontology elements is illustrated in figure 6. A client, e.g. the annotation client, initiates a search by passing a keyword

[14]An introduction to the SPARQL query language for RDF can be found at http://www.w3.org/TR/rdf-sparql-query/

to the *Semantic Search Service*. The *Semantic Search Service* constructs one or more SPARQL queries and sends them to the *Repository Service* by which they are evaluated. The result of each query is sent back to the *Semantic Search Service* that constructs the result set for the client. Finally, the found elements are returned to the client.

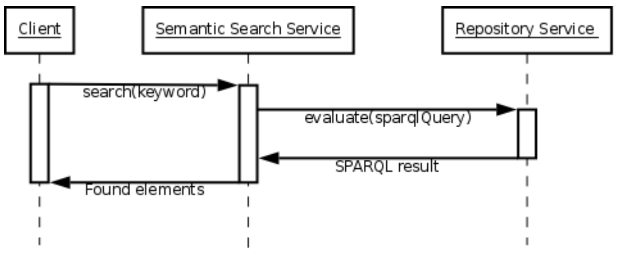


Figure 6: Semantic Search Sequence Diagram

In the following we detail the search strategy.

Search Strategy:

1. Ontology elements are retrieved on the basis of string matching; the given search keyword is compared to the names of the ontology elements in the repository. The used string matching strategies are listed below:

    - Exact string matching.
    - Starts-with string matching.
    - Sub-string matching.

    The different strategies are needed for the ranking of the found elements as described below.

2. Sub-elements for each element found by string matching are retrieved. Thereby, not only direct sub-elements are considered but also sub-elements of sub-elements until the leafs of the hierarchy or the maximal allowed depth, that can be specified, are reached. This part of

the search is the semantic part because it adds elements which are not found by string matching but with a meaning related to the elements found by string matching to the search result.

3. The found elements are sorted due to its ranking, arranging elements with a high ranking at the beginning and elements with a low ranking ranking at the end of the list. The following classification groups are used for the ranking:

    - *High ranking:* Elements found by exact string matching (and its sub-elements) have a high ranking.
    - *Middle ranking:* Elements found by starts-with string matching (and its sub-elements) have a middle ranking.
    - *Low ranking:* Elements found by sub-string matching (and its sub-elements) have a low ranking.

    For elements within the same ranking group there is no specific ordering defined, except that sub-elements are arranged after its parent element.

The search algorithm in pseudocode is depicted in figure 7 and 8.

```
SEARCH(keyword)
INPUT       The search keyword
OUTPUT      The list of found elements. The elements with high ranking are
            arranged at the top of the list and the elements with low ranking
            a the bottom of the list.

// Initialize the array containing the used string-matching strategies.
// Elements found by stragegies with lower index get a higher ranking.
strategy[0] := EXACT_MATCHING // high ranking
strategy[1] := STARTS_WITH_MATCHING // middle ranking
stragegy[2] := SUB_STRING_MATCHING // low ranking

// Initialize an empty list for the found elements
result := empty list

// Set the max depth for sub-elements
depth := 1

// Search for elements using each of the defined search strategies
for i=0 to 2 do
    foundElements := elements found by stragegy strategy[i]
    for each e in foundElements do
        if e not in result then
            Add e to the end of the result list
            ADD_SUB_ELEMENTS(e, result, depth)

return result
```

Figure 7: Search Strategy - Part 1

```
ADD_SUB_ELEMENTS(element, result, depth)
    INPUT        Element, result list, and max depth

    if depth > 0 then
        subElements := list of sub-elements of element
        for each e in subElements do
            if e not in result then
                Add e to the end of the result list
                depth := depth - 1
                ADD_SUB_ELEMENTS(e, result, depth)
```

Figure 8: Search Strategy - Part 2

Usage Example:
Due to this search strategy we present an example of a concept search initiated using the keyword *c*. Considering that the only ontology stored in the repository is the one depicted in figure 9 the search result is the following:

- Conference *(middle ranking)*
- Object *(low ranking)*
- Book *(low ranking, sub-element of Object)*
- Person *(low ranking, sub-element of Object)*
- OlympicGames *(low ranking)*

The concepts *Conference*, *Object*, and *OlympicGames* are directly found by string matching. *Conference* is at the top of the list because the starts-with string matching has a higher ranking as the sub-string matching, which is used to find the concepts *Object* and *OlympicGames.* The concepts *Book* and *Person* are sub-concepts of *Object* and, therefore, they have the same ranking as its parent concept and are arranged directly after it in the list.

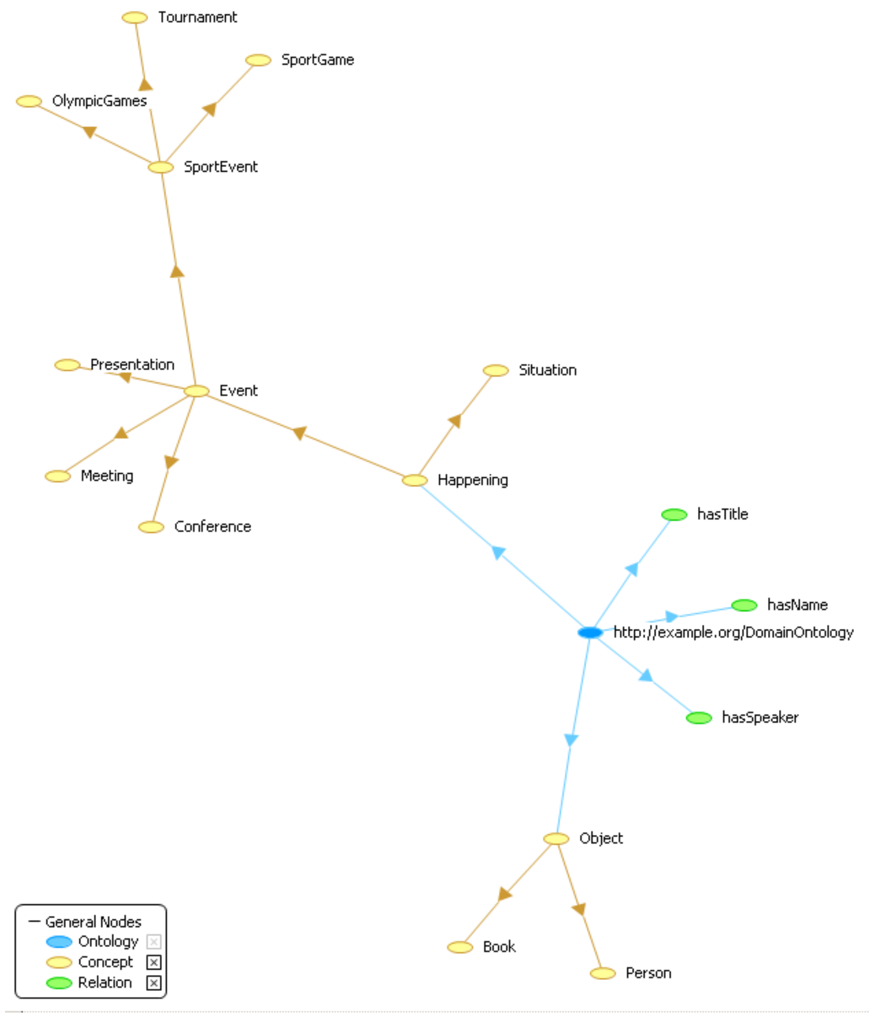


Figure 9: Example Ontology

2.3.4 Ontology Extension (Element Addition) Service

The *Ontology Extension Service* offers functionalities for the realization of the *Element Addition* part of the SA Methodology. It provides functions to add new concepts and relations to an ontology stored in the repository as well as functions for the realization of the manual classification approach of the SA Methodology.

Ontology elements added by the help of this service are all added to the same ontology, namely the *Extension Ontology.* It's an ontology used to group all the concepts and relations added by the user. This design decision was taken, because these elements can contain spelling or classification errors. According to [3] elements added by users mature over time by being used and modified. Once elements reach a certain level of maturity, they can be included by domain and ontology experts into one of the other ontologies stored in the repository.

Based on the methods of the *Ontology Extension Service* the manual classification support of new elements, presented in [1], can be realized. This classification approach guides the user through the elements of the used ontologies by asking to define the new element to be a sub-element of an existing one. Firstly, a list of root elements of the used ontologies is presented to the user that can select one of them in order to classify the new element. Each time when an element is selected, its sub-elements are presented to the user that can choose one of them to further specialize the classification. The *Ontology Extension Service* provides methods for the retrieval of root elements of ontologies as well as sub-element of an element, allowing the realization of this classification approach. A detailed description of the interface of the *Ontology Extension Service* is shown in appendix A.4.

2.4 Annotation Client

The annotation client described in this section is based on the functionalities of the annotation framework presented in the previous section. It provides a graphical user interface for manual annotation of resources stored on a local harddisk, including features to support users in the selection of ontology elements needed for annotation as well as the possibility to add missing ontology elements during annotation time.

2.4.1 User Interface Mockups

Figure 10 on page 23 depicts the main interface of the client representing the scenario of a resource being annotated on the basis of the example depicted in figure 4 on page 14. The areas of the user interface labeled with numbers in red are described in the following:

1 Drop-down menu providing access to some main functionalities such as loading resources or storing annotations.

2 Area displaying the location of the loaded resource.

3 Selection field for defining the type of the resource being annotated. There are five possible choices for the resource type that correspond to concepts of the annotation ontology presented in section 2.3.2 (see table 2). So for example if the type *Image* is selected, the annotation is defined to be an *ImageAnnotation*.

Type	**Concept**
Image	ImageAnnotation
Audio	AudioAnnotation
Video	VideoAnnotation
Text	TextAnnotation
Other	Annotation

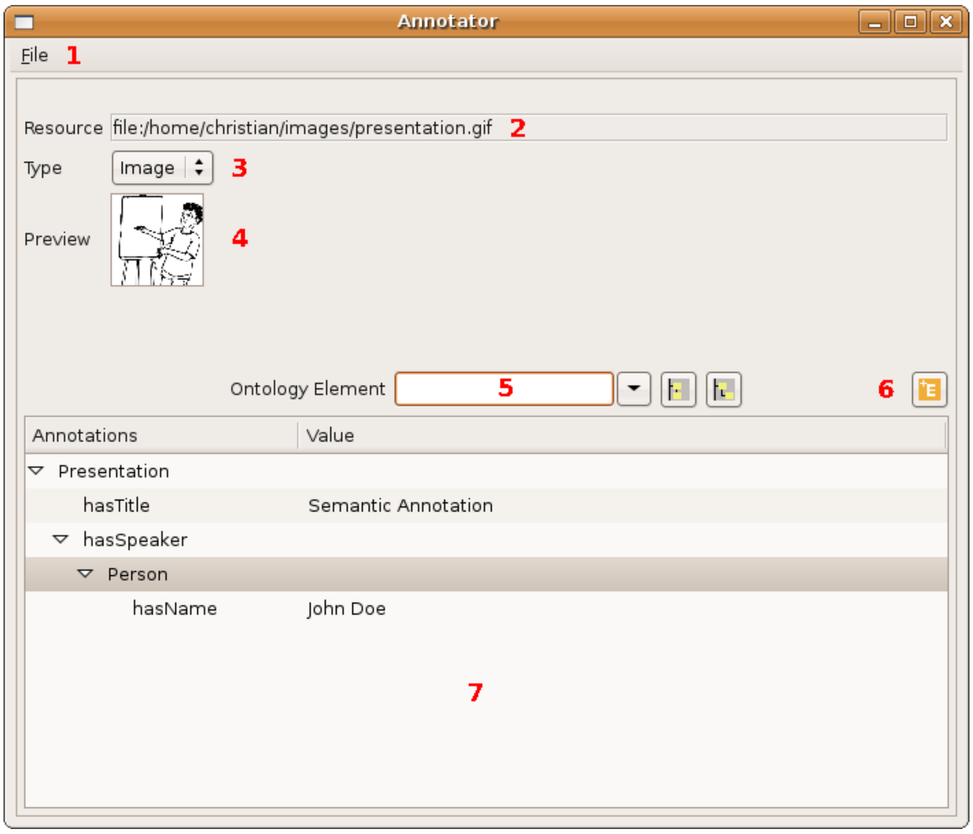
Table 2: Resource Types

4 Preview of the loaded resource. This functionality is provided only for image resources. A click on the preview opens a popup window with a larger preview. If the resource type is not an image the preview is empty.

5 Field for searching for ontology elements that provides semantic auto-completion capabilities. The found elements can be used for annotating the loaded resource. A more detailed description about this part of the user interface is provided in section 2.4.2.

6 Button for adding new ontology elements. A more detailed description about this part of the user interface is given in section 2.4.2.

7 Area displaying annotations about resources. Annotations are organized as a tree structure offering the possibility to construct complex descriptions. A more detailed presentation about the creation of annotations is provided in section 2.4.2.



Resource file:/home/christian/images/presentation.gif

Type Image

Preview

Ontology Element

Annotations	Value
Presentation	
hasTitle	Semantic Annotation
hasSpeaker	
Person	
hasName	John Doe

Figure 10: Annotation Client GUI

2.4.2 Realization of the Features of the Annotation System in the Annotation Client

The realization of the features for manual annotation of resources and for the support of the user in the execution of this task in the annotation client are described in this section.

Semantic Search with Autocompletion
Semantic search with autocompletion is supported in the user interface in the area labeled with 5 in figure 10. The idea of this part is that the user types a word she wants to use for the creation of annotations of the loaded resource and a list of available concepts and relations related to the inserted keyword is presented to the user. The elements of the list can be used to annotate resources.

For the generation of the list the *Semantic Search Service* as presented in section 2.3.3 is used. Thereby, while the user is typing, search requests are triggered and the results are presented to the user. Figure 11 illustrates a search scenario with a user typing the word *ev* and the list of found concepts presented in a popup. The prefix *ns1* of the found elements identifies the namespace of the elements. In this case all elements are within the same namespace. The elements stored in the repository are the ones of the ontology depicted in figure 9 on page 20.

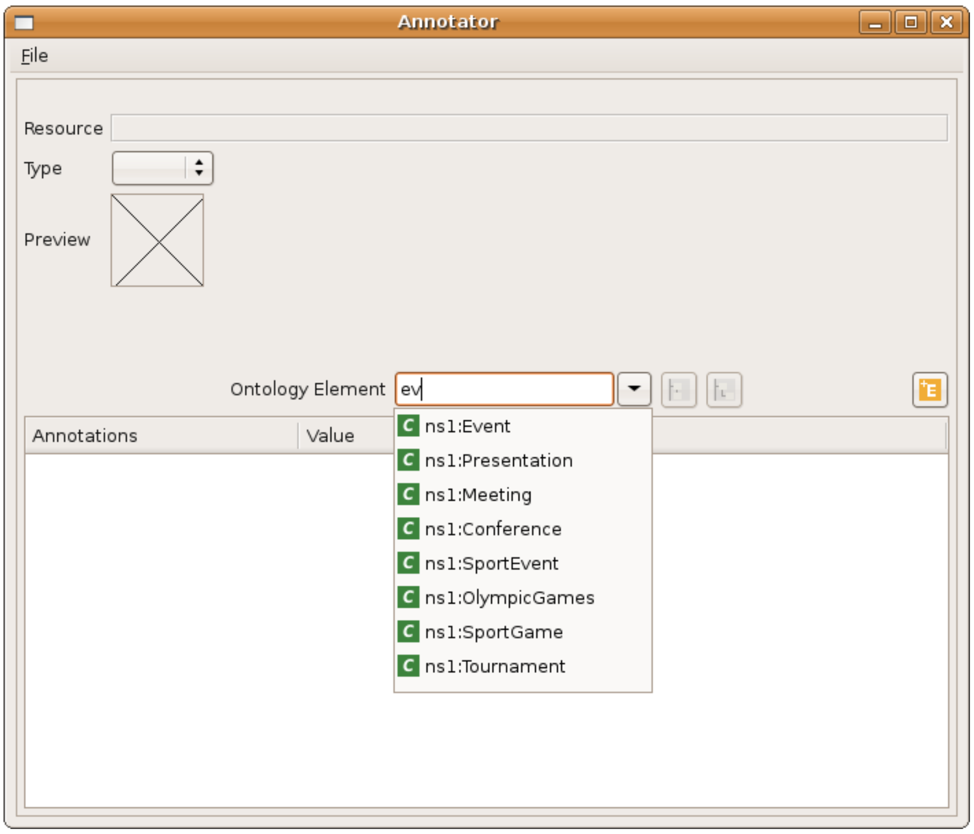


Figure 11: Element Search - Example 1

Figure 12 illustrates another example of an element search with the user typing the word *k*. In this example the result is the relation *hasSpeaker* and the concept *Book* of our example ontology being returned.

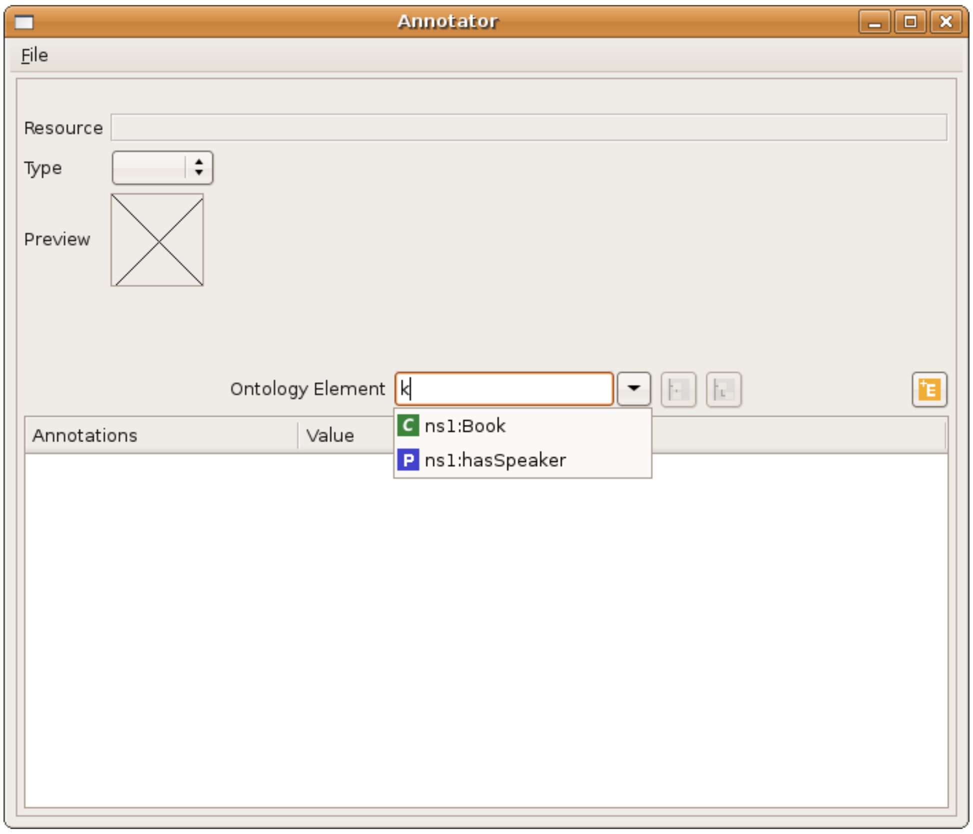


Figure 12: Element Search - Example 2

For cases in which the search result is empty the annotation client provides the possibility to use the inserted word anyway, by adding it as a new ontology element to the repository. This feature is described in the next section.

Element Addition

The *Element Addition* part of the user interface provides the functionality to add missing concepts and relations to the ontology elements stored in the repository. It realizes the *Element Addition* part of the SA Methodology including manual support for the classification of newly added elements. For the implementation of this component the *Ontology Extension Service* presented in section 2.3.4 is used.

Figure 13 shows a situation of a user typing the word *Concert*. The list of results is empty because there are no ontology elements which correspond to the inserted word. An extension of the search strategy to find synonyms of the inserted word could improve the search result in such cases. The user can decide to add *Concert* as a new element to the ontology by pushing the orange button labeled with *E* to the right of the search field. Once the button is pushed, a new window pops up providing a wizard for the addition of the new element and guiding the user through the process of manual classification of the element as described in section 2.3.4.

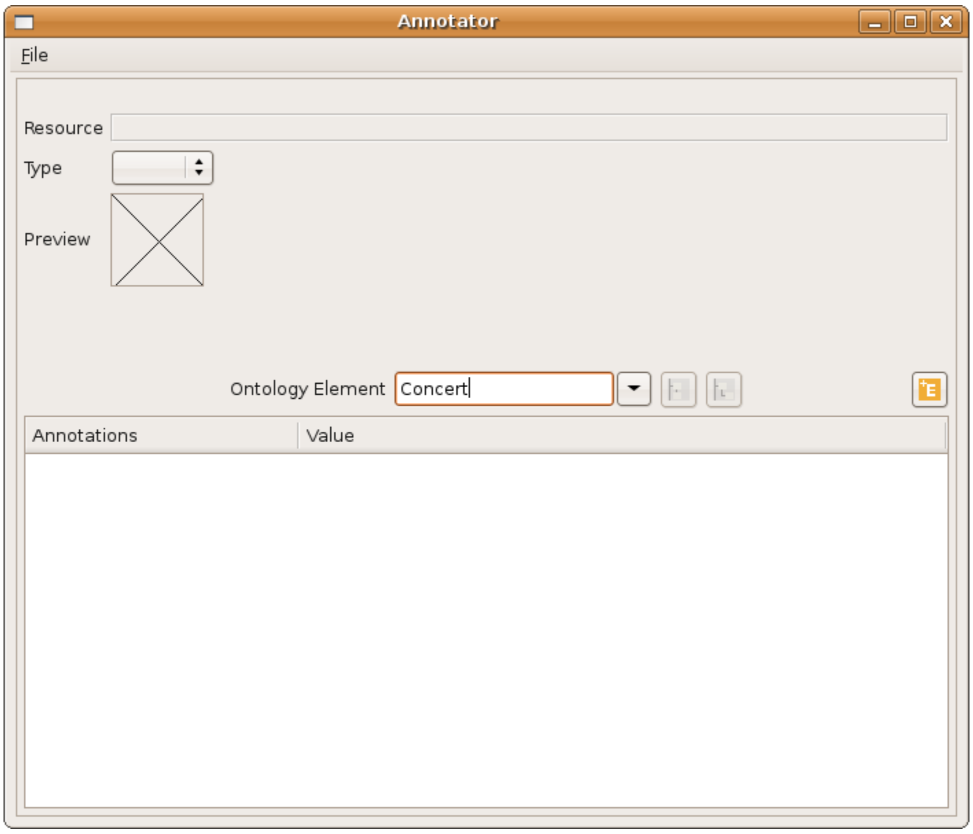


Figure 13: Element Addition Example - Step 1

Figure 14 depicts the first page of the element addition wizard where the name of the new ontology element can be specified. Moreover, the user has to decide weather the element should be a concept or a relation. The addition of instances, representing a further possibility for addition of ontology elements, is not supported by the annotation system.

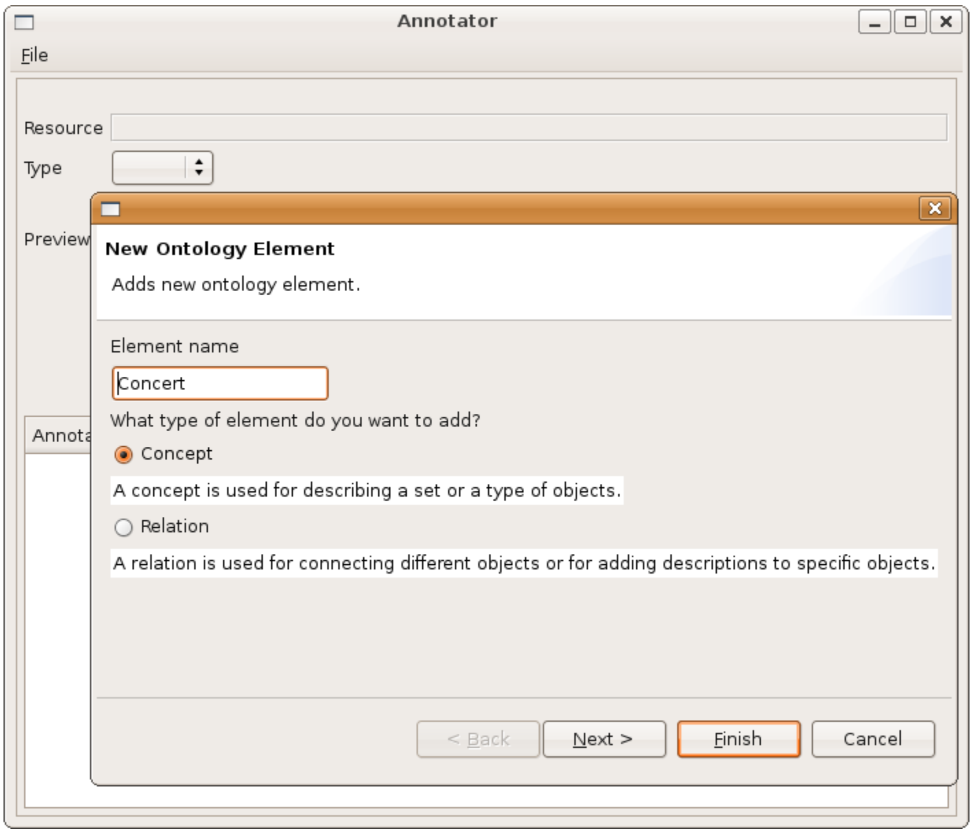
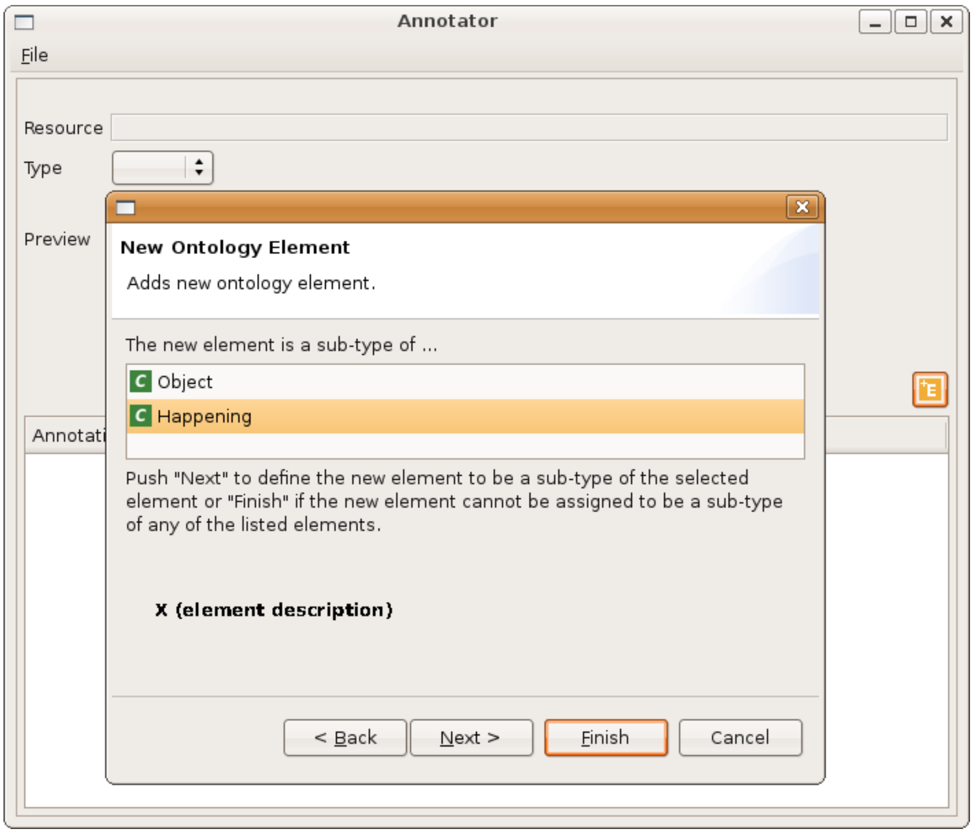

Annotator
File
Resource
Type
Preview
Annota
New Ontology Element
Adds new ontology element.
Element name
Concert
What type of element do you want to add?
Concept
A concept is used for describing a set or a type of objects.
Relation
A relation is used for connecting different objects or for adding descriptions to specific objects.
< Back
Next >
Finish
Cancel

Figure 14: Element Addition Example - Step 2

Once the name and the type of the new ontology element are specified, the classification process of the new element begins. This process guides the user through the elements of the domain ontologies which are stored in the repository by asking for classification of the element as a sub-concept or sub-relation of existing ones. The new element can also be defined as a root element. Figure 15 presents the screen of the classification support. On this wizard page the root elements of the domain ontologies stored in the repository are presented. In our example, the only ontology which is stored in the repository is the one of figure 9 on page 20, and, therefore, the displayed elements are *Happening* and *Object*[15]. These elements are generic enough in order to group lots of new elements added by users. The visualization of a description of the selected ontology element, for example, in the area marked with *X* in figure 15, could help the user in the selection of an adequate element. This feature is not implemented in the annotation system. In our example the user defines the new element *Concert* to be a *Happening*.


Annotator

File

Resource

Type

Preview

Annotati

New Ontology Element

Adds new ontology element.

The new element is a sub-type of ...

Object

Happening

Push "Next" to define the new element to be a sub-type of the selected element or "Finish" if the new element cannot be assigned to be a sub-type of any of the listed elements.

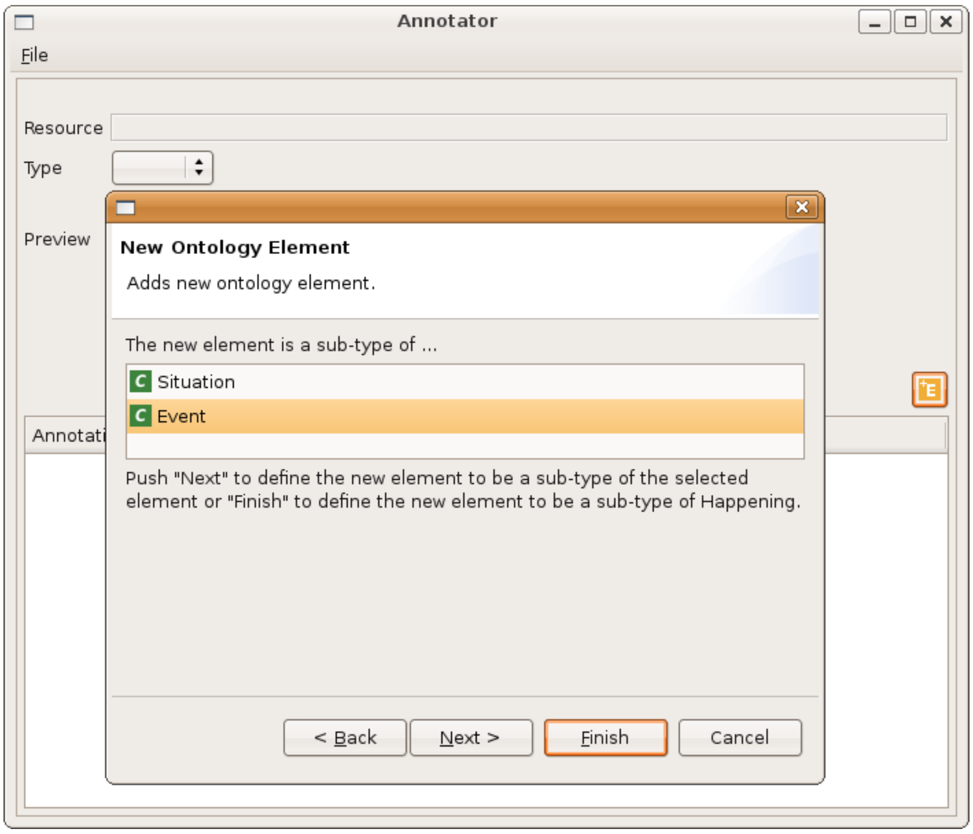
X (element description)

Figure 15: Element Addition Example - Step 3

On the next page depicted in figure 16 the user can specialize the classification by defining the new element to be an *Event* or a *Situation*. These two concepts are the sub-concepts of *Happening*. Its also possible to decide

[15]This example ontology uses some concepts of the PROTON ontology and *Happening* and *Object* are two of those (http://proton.semanticweb.org/)

that *Concert* is neither an *Event* nor a *Situation*. In this case *Concert* would be defined to be a *Happening*. In our case the user decides that *Concert* is an *Event*.



New Ontology Element

Adds new ontology element.

The new element is a sub-type of ...

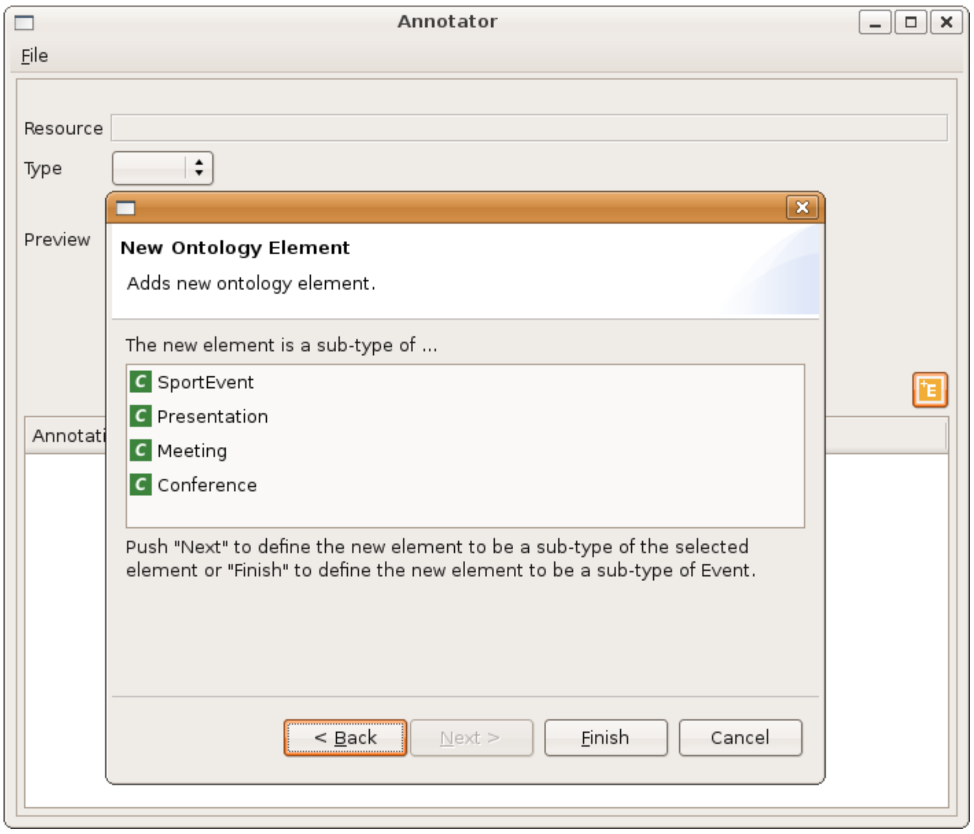
Situation

Event

Push "Next" to define the new element to be a sub-type of the selected element or "Finish" to define the new element to be a sub-type of Happening.

Figure 16: Element Addition Example - Step 4

In figure 17 the next step of the classification process is depicted. The sub-concepts of *Event* are presented and the user is asked to further specialize the classification. In our case the classification of *Concert* is not further specialized and it is defined to be an *Event*.



Push "Next" to define the new element to be a sub-type of the selected element or "Finish" to define the new element to be a sub-type of Event.

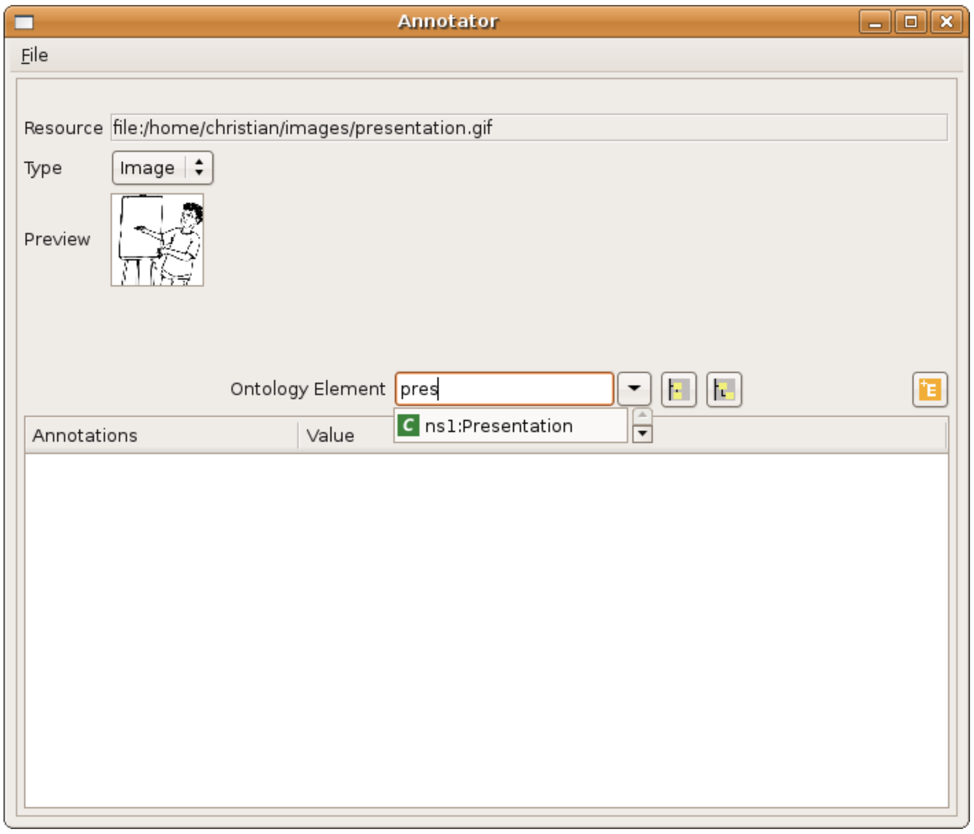
Figure 17: Element Addition Example - Step 5

Now that the new element *Concert* is added to the repository, it can be used for the annotation of resources. The mechanism for creating annotations is presented in the next section.

Annotation Management

In this section we present the process of creating annotations about resources enabled by the user interface of the annotation client. Thereby, we illustrate how the annotation of figure 4 on page 14 can be created. The first step for the creation of annotations to a resource is the selection of ontology elements to use.

Figure 18 shows the user interface with the image depicting someone giving a presentation loaded. The tree representing the annotations is empty because there are no annotations to the resource yet. The user wants to specify that the image is about a presentation and inserts the word *pres* into the ontology element search field. The autocompletion capability presents the concept *Presentation* and the user creates an annotation using this concept by pushing the “Add” button to the right of the search field.

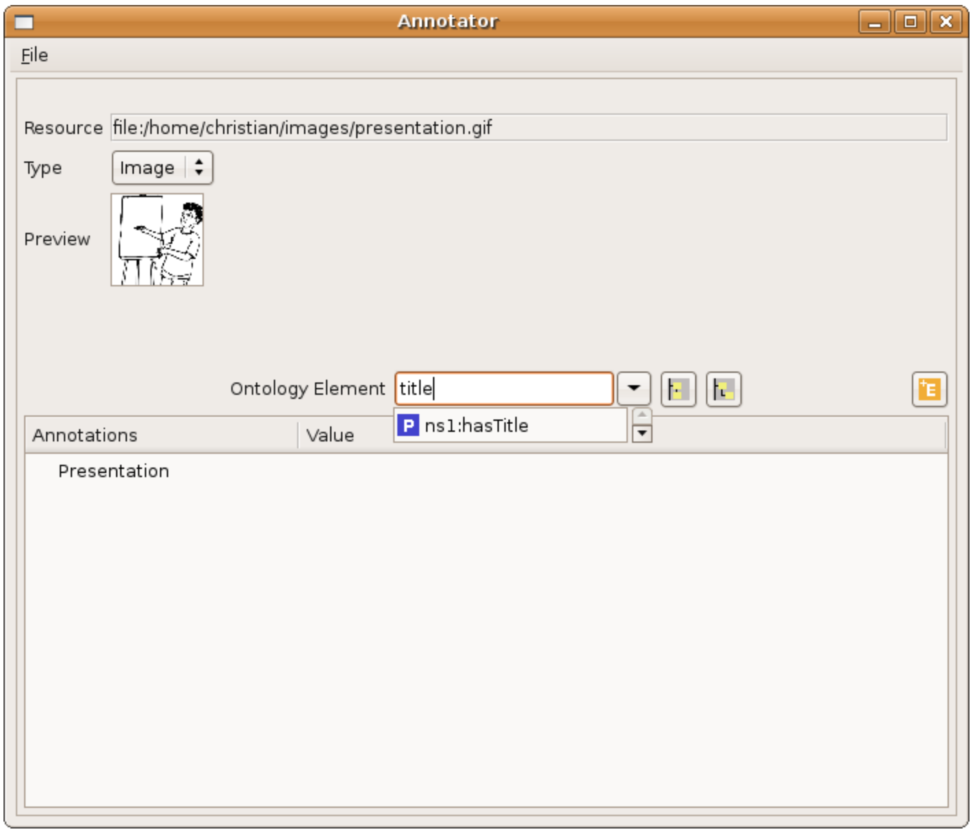


Resource: file:/home/christian/images/presentation.gif

Ontology Element: pres

Figure 18: Annotation Creation Example - Step 1

In figure 19 we see that the concept *Presentation* is added to the annotation tree as a top level element. Now the user wants to specify the title of the presentation and inserts the word *title* into the search field. The relation *hasTitle* is presented to the user that utilizes it. For the addition of children to elements of the tree, such as *hasTitle* as child of *Presentation*, first the tree element *Presentation* has to be selected and then the second button to the right of the search field has to be used. Once the button is pushed a new window pops up where the user can complete the creation of this annotation.



Resource file:/home/christian/images/presentation.gif

Type Image

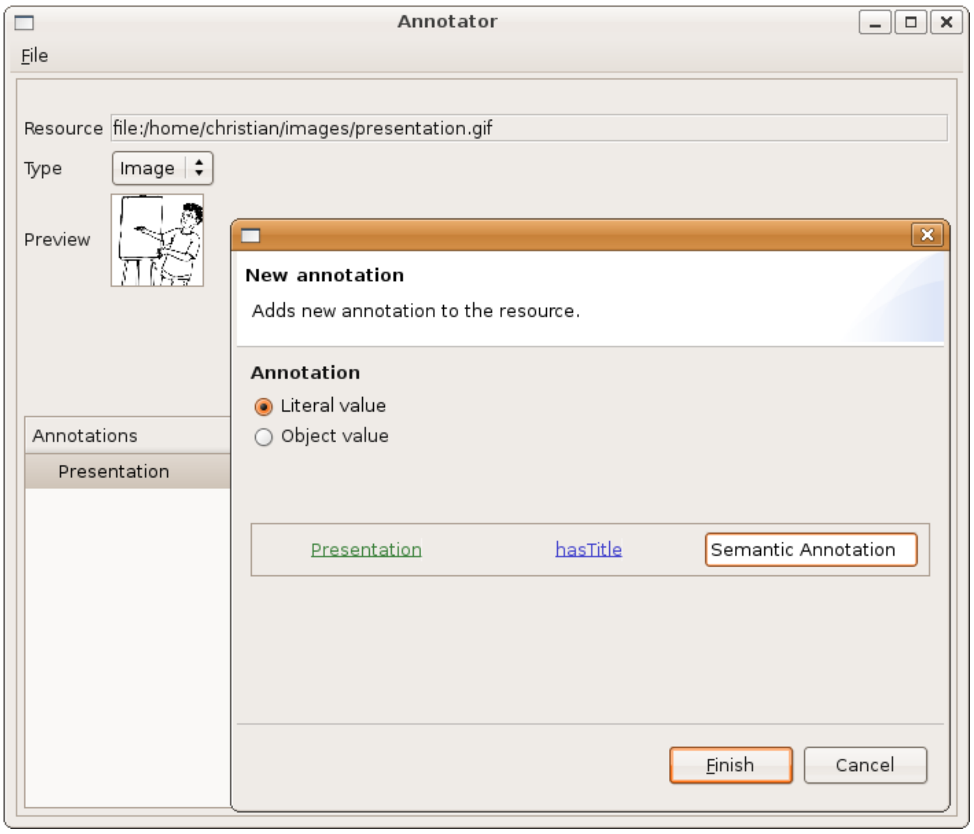
Ontology Element title

ns1:hasTitle

Presentation

Figure 19: Annotation Creation Example - Step 2

Figure 20 depicts the popup window of the annotation client that offers the possibility to insert a value for the selected relation. Thereby, the user can decide between a litaral value and an object value. The former selection provides an input field where the user can freely type a string that represents the value of the relation. The latter provides a selection field with the concepts of the domain ontologies which are stored in the repository. The user can select one of the elements in the field as the value of the relation. In our example a literal value is used.



New annotation

Adds new annotation to the resource.

Annotation

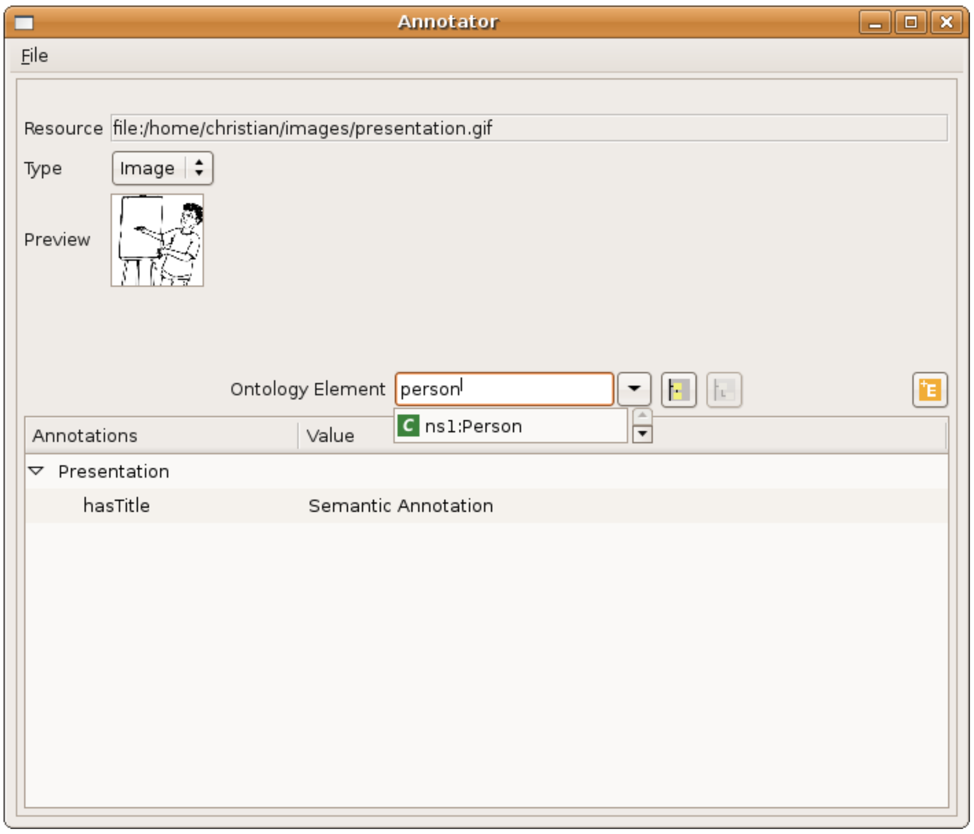
[✓ Literal value]

Object value

Presentation hasTitle Semantic Annotation

Figure 20: Annotation Creation Example - Step 3

In the next step depicted in figure 21 the user wants to extend the information about the event represented in the picture by specifying the person that gives the presentation and types the word *person*. The autocompletion capability presents the concept *Person* as the result. The user creates the annotation by using this concept. Afterwards, the popup window for completing the annotation opens again.

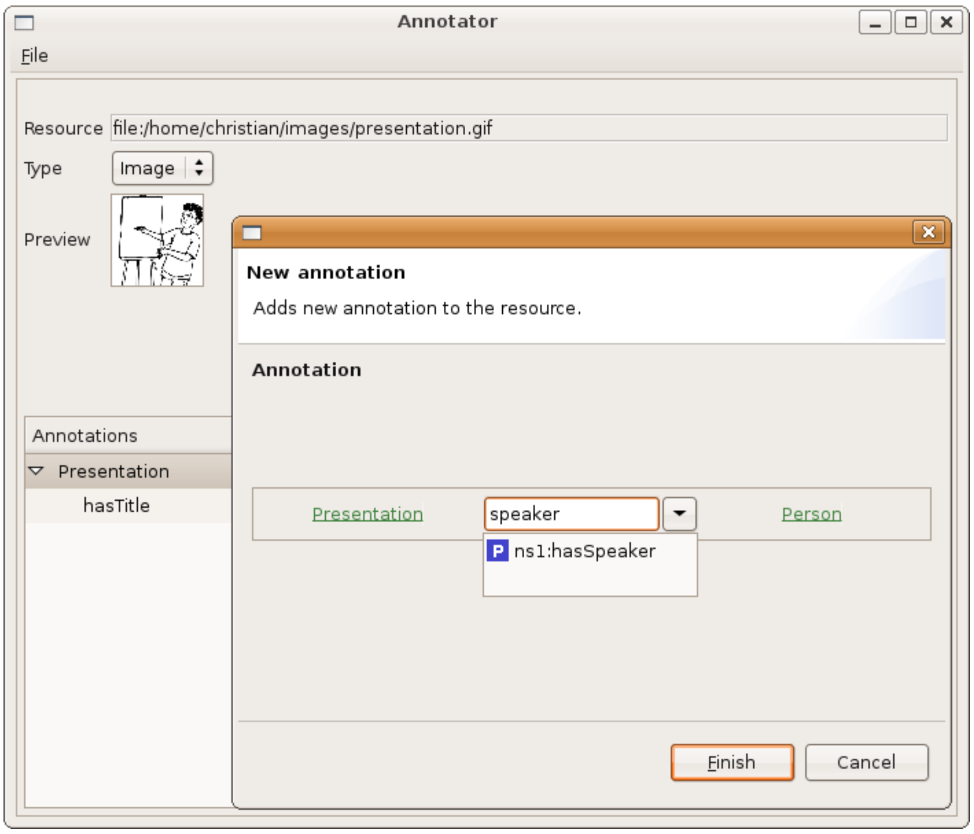


file:/home/christian/images/presentation.gif

Presentation
hasTitle Semantic Annotation

Figure 21: Annotation Creation Example - Step 4

Figure 22 depicts the annotation popup window for the addition of the concept *Person* to the annotation tree. In this case the user has to specify how *Person* is related to *Presentation* by selecting one of the relations provided by the domain ontologies and decides to utilize the relation *hasSpeaker* for this connection. As soon as this relation is added to the annotation tree, the name of the person is added in the same way as the title of the presentation presented before resulting in the annotation tree depicted in figure 10 on page 23.



Resource file:/home/christian/images/presentation.gif

Figure 22: Annotation Creation Example - Step 5

On the basis of this process arbitrary annotations about resources can be created. Once the descriptions are complete, the user can store them in the repository by using the *Save* functionality which is provided by the *File* menu.

3 Conclusions

In this thesis we presented an annotation system enabling ontology-based annotation of resources based on the SA Methodology [1]. The functionality of the SA Methodology is realized by the annotation framework and the annotation client. The former consists of a set of Web services providing the necessary functionalities for the management of annotations as well as capabilities for a partly realization of the *Element Selection* and *Element Addition* parts of the SA Methodology. The latter, using the capabilities of the annotation framework, represents the application for end users enabling manual creation of ontology-based metadata about resources stored on a local harddisk. An easy retrieval of ontology elements is provided on the basis of the semantic search with autocompletion. Furthermore, the addition of ontology elements is possible during annotation time.

The *Semantic Search with Autocompletion*, the *Ontology Extension*, and the *Classification Support* components of the SA Methodology are realized in this thesis. The first component belongs to the *Element Selection* part of the methodology and is realized by the implementation of semantic search based on the hyponym relation among ontology elements. This approach provides search results containing ontology elements not only being directly found by string matching but also concepts related to the directly found ones by being its hyponyms. A possible extension of the search mechanism provided by the framework could be the integration of WordNet[16] into the search process [10]. WordNet is a database of over 150 thousand words of the English language including connections among them such as synonym or hyponym relations. By the use of vocabulary which is provided in this lexicon synonyms or hyponyms of keywords inserted by users could be retrieved. These additional keywords could be integrated into the search query constructed by the *Semantic Search Service.*

The *Element Selection* part of the methodology is realized by the *Ontology Extension* and the *Classification Support* components. The latter is realized on the basis of the manual classification approach which is presented in [1] and supports the user in the classification of newly added ontology elements. The former component provides the possibility to add new ontology elements to the available vocabulary. However, modifications of these added elements are not possible. According to [3] the possibility of modifying newly added elements would be important in order to improve the quality of the

[16]http://wordnet.princeton.edu/

vocabulary over time.

References

[1] Christian Ammendola. A manual annotation approach based on ontologies, March 2008. URL `http://www.sti-innsbruck.at/teaching/theses/completed/details/?uid=94`. (Bachelorthesis).

[2] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284:34–43, 2001.

[3] Simone Braun, Andreas Schmidt, and Andreas Walter. Ontology maturing: a collaborative web 2.0 approach to ontology engineering. In *Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge at the 16th International World Wide Web Conference (WWW 07), Banff, Canada*, 2007. URL `http://www.andreas-p-schmidt.de/publications/ontology_maturing_braun%_schmidt_walter_www07.pdf`.

[4] Jos de Bruijn, Holger Lausen, Reto Krummenacher, Axel Polleres, Livia Predoiu, Michael Kifer, and Dieter Fensel. D16.1v0.2 the web service modeling language wsml, March 2005.

[5] Marin Dimitrov, Vassil Momtchev, Alex Simov, Damyan Ognyanoff, and Mihail Konstantinov. wsmo4j programmers guide. Technical report, OntoText Lab, November 2006. URL `http://wsmo4j.sourceforge.net`.

[6] Cristina Feier and John Domingue. D3.1v0.1 wsmo primer, April 2005.

[7] Thilo Frotscher, Marc Teufel, and Dapeng Wang. *Java Web Services mit Apache Axis2.* entwickler.press, 2007.

[8] Asuncion Gomez-Perez, Oscar Corcho-Garcia, and Mariano Fernandez-Lopez. *Ontological Engineering.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003. ISBN 1852335513.

[9] Siegfried Handschuh and Steffen Staab. *Annotation for the Semantic Web*. IOS Press, 2003.

[10] Laura Hollink, Guus Schreiber, and Bob Wielinga. Patterns of semantic relations to improve image content search. *Web Semant.*, 5(3):195–203, 2007.

[11] Cay S. Horstmann and Gary Cornell. *Core Java: Volume I, Fundamentals*. Prentice Hall, 8 edition, September 2007.

[12] José Kahan, Marja-Riitta Koivunen, Eric Prud'Hommeaux, and Ralph R. Swick. Annotea: An open rdf infrastructure for shared web annotations, May 2001.

[13] Jeff McAffer and Jean-Michel Lemieux. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications*. Addison-Wesley Professional, 2005.

[14] Vassil Momtchev and Atanas Kiryakov. Ordi: Ontology representation and data integration framework, October 2006.

A Annotation Services (Interfaces)

The annotation framework presented in section 2.3 consists of a set of Web services. A detailed description of the interfaces of these services is provided in the subsequent sections.

A.1 Repository Service Interface

The *Repository Service* interface provides a set of methods for the management of WSMO elements stored in the repository. Figure 23 depicts the interface of the service and an explanation of the methods is provided by the following list:

- *saveX:* The *save* methods store WSMO element in the repository. If the element already exists it is removed before the new version of the element is stored. WSMO elements are passed to the methods as a string in WSML human-readable syntax.
- *addX:* The *add* methods store WSMO elements in the repository. They do not remove the object from the repository before adding the new one if it already exits. WSMO elements are passed to the methods as a string in WSML human-readable syntax.
- *getX:* The methods beginning with *get* return WSMO elements. The return value is a WSMO element as a string in WSML human-readable syntax.
- *listX:* The methods beginning with *list* return the identifier of WSMO elements.
- *deleteX, removeX:* The *delete* and *remove* methods remove WSMO elements from the repository.
- *removeAllX:* The removeAll methods removes the specified WSMO element and all its associated elements recursively. The method supports the removal of the following WSMO elements:

- *Concepts:* Removes the concept and all its instances.
- *Relation:* Removes a relation and all its instances.
- *Instance:* Removes an instance with all its attributes and all relation instances using the instance.
- *Relation Instance:* Removes the relation instance.

• *evaluate:* Executes a SPARQL query and returns the results in SPARQL xml result format.

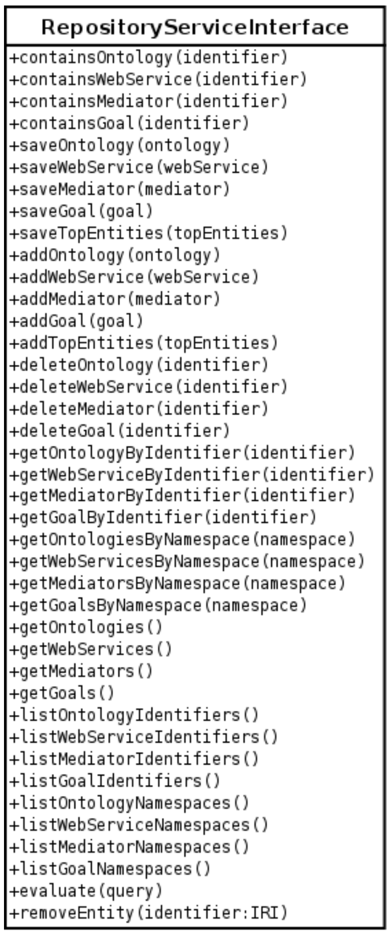

RepositoryServiceInterface
+containsOntology(identifier)
+containsWebService(identifier)
+containsMediator(identifier)
+containsGoal(identifier)
+saveOntology(ontology)
+saveWebService(webService)
+saveMediator(mediator)
+saveGoal(goal)
+saveTopEntities(topEntities)
+addOntology(ontology)
+addWebService(webService)
+addMediator(mediator)
+addGoal(goal)
+addTopEntities(topEntities)
+deleteOntology(identifier)
+deleteWebService(identifier)
+deleteMediator(identifier)
+deleteGoal(identifier)
+getOntologyByIdentifier(identifier)
+getWebServiceByIdentifier(identifier)
+getMediatorByIdentifier(identifier)
+getGoalByIdentifier(identifier)
+getOntologiesByNamespace(namespace)
+getWebServicesByNamespace(namespace)
+getMediatorsByNamespace(namespace)
+getGoalsByNamespace(namespace)
+getOntologies()
+getWebServices()
+getMediators()
+getGoals()
+listOntologyIdentifiers()
+listWebServiceIdentifiers()
+listMediatorIdentifiers()
+listGoalIdentifiers()
+listOntologyNamespaces()
+listWebServiceNamespaces()
+listMediatorNamespaces()
+listGoalNamespaces()
+evaluate(query)
+removeEntity(identifier:IRI)

Figure 23: Repository Service Interface

A.2 Annotation Service Interface

The *Annotation Service* interface provides methods for the management of annotations. Figure 24 depicts the interface of the service and a description of the methods can be found in the following:

- *getDomainOntologies:* This method returns the domain ontologies stored in the repository. Domain ontologies are the ontologies describing a certain domain which elements can be used for describing resources in annotations. The return value is a string representing the ontologies in WSML human-readable syntax.
- *saveAnnotation:* Stores an annotation to the repository. If an annotation about the same resource of the new annotation already exists, then it is overwritten. Annotations are added to a separate ontology named *Annotations* and not to the annotation ontology or to a domain ontology.
- *getAnnotation:* Loads annotations about resources.
- *removeAnnotation:* Removes annotations about resources.

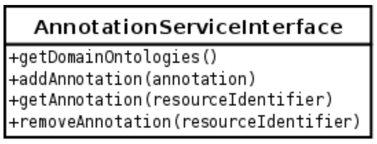

AnnotationServiceInterface
+getDomainOntologies()
+addAnnotation(annotation)
+getAnnotation(resourceIdentifier)
+removeAnnotation(resourceIdentifier)

Figure 24: Annotation Service Interface

A.3 Semantic Search Service Interface

The *Semantic Search Service* interface provides a set of methods for the retrieval of concepts and relations. The description of the methods of the interface depicted in figure 25 can be found in the following:

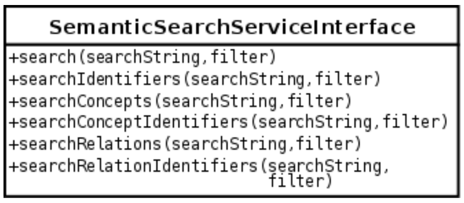

SemanticSearchServiceInterface
+search(searchString,filter)
+searchIdentifiers(searchString,filter)
+searchConcepts(searchString,filter)
+searchConceptIdentifiers(searchString,filter)
+searchRelations(searchString,filter)
+searchRelationIdentifiers(searchString, filter)

Figure 25: Semantic Search Service Interface

The interface provides methods for searching concepts and relations that can be divided into two groups:

- *searchXIdentifiers:* These methods search for ontology elements matching the search string and return the identifiers of the found elements. The filter parameter provides the possibility to specify a set of namespaces not consideres in the search.
- *searchX:* Methods not ending with *Identifiers* search for ontology elements matching the search string and return the elements itself. The filter parameter provides the possibility to specify a set of namespaces not consideres in the search.

A.4 Ontology Extension Service Interface

The *Ontology Extension Service* interface offers methods for the realization of the *Element Addition* part of the SA Methodology. Figure 26 presents the interface of the service and an in detail description of its methods can be found in the following:

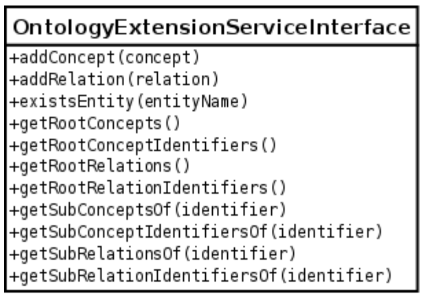
- *addX:* Adds new concepts or relations to the repository. The elements are added to an ontology used for grouping all the elements added by this service, namely the *Extension Ontology.*
- *containsEntity:* Checks if an *Entity* with a specified name already exists in the set of elements of the extension ontology.

- *getRootX:* Returns the root elements of all domain ontologies stored in the repository. Root elements are elements not having a parent element.

- *getRootXIdentifiers:* Returns the identifier of all root elements being part of the domain ontologies stored in the repository.

- *getSubXOf:* Returns the sub-elements of the specified element.

- *getSubXIdentifiersOf:* Returns the identifier of the sub-elements of the specified element.



```
OntologyExtensionServiceInterface
+addConcept(concept)
+addRelation(relation)
+existsEntity(entityName)
+getRootConcepts()
+getRootConceptIdentifiers()
+getRootRelations()
+getRootRelationIdentifiers()
+getSubConceptsOf(identifier)
+getSubConceptIdentifiersOf(identifier)
+getSubRelationsOf(identifier)
+getSubRelationIdentifiersOf(identifier)
```

Figure 26: Ontology Extension Service Interface

B Annotation Framework (Packages and Deployment)

The annotation framework is realized base on Axis2. For the implementation we used the version 1.3 of the framework.

Service Packages

The code of each annotation service is organized in three packages listed in the following (the *XX* in the package names identifies the respective service: *repository*, *annotation*, *semanticsearch*, or *ontologyextension*):

- *at.sti2.services.XX.server*:
  This package contains the server side code of the annotation service. It includes the WDSL specification of the Web service and the implementation of the Web service interface.

- *at.sti2.services.XX.client*:
  This package contains the client side code of the annotation service. This client can be used in order to access the functionalities of the Web service.

- *at.sti2.services.XX.common*:
  This package contains code used by both, the server and the client side of the annotation service.

Packaging

For the packaging of the code of the annotation framework we used Maven2[17]. For each annotation service the following archives are created:

- *XXService.aar*:
  This archive includes the code of the Web service and all the necessary libraries. This archive can be deployed on an Axis2 server.

- *XX-common-VERSION.jar*:
  This archive contains the code of the common package.

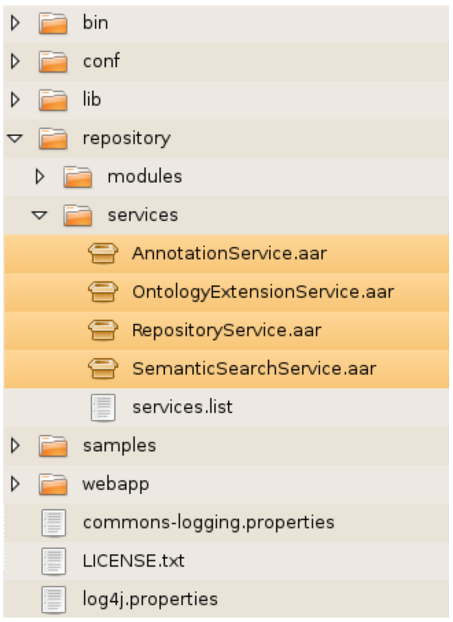
- *XX-client-VERSION.jar*: This archive contains the code of the client package.

Deployment

For the deployment of the Web services we used the standalone server included in the Axis2 distribution. Figure 27 depicts the folder structure of the server. The aar-archives of the Web services of the annotation framework are situated in the *repository/services/* folder of this structure and the

[17]http://maven.apache.org/

bin/ folder includes the *axis2server.sh* and *axis2server.bat* scripts used for starting the server.



```
▷ bin
▷ conf
▷ lib
▽ repository
    ▷ modules
    ▽ services
        AnnotationService.aar
        OntologyExtensionService.aar
        RepositoryService.aar
        SemanticSearchService.aar
        services.list
▷ samples
▷ webapp
  commons-logging.properties
  LICENSE.txt
  log4j.properties
```

Figure 27: Axis2 Server

C Annotation Client (Packages)

The annotation client implementation is based on the Eclipse RCP framework. The application has the following package structure:

- *at.sti2.annotationclient.model*:
  This package contains the code accessing the functionalities of the annotation framework. It uses the code provided by the *XX-client-VERSION.jar* archives for accessing the Web services.

- *at.sti2.annotationclient.ui*:
  This package contains the graphical elements of the application. It uses the model for accessing the functionalities of the Web services.

- *at.sti2.annotationclient*:
  This package contains the main plug-in of the annotation client. It contains the code that is executed in order to start the application.