Public Web Services for Ontology Library Systems

Ville-Pekka Komulainen

Helsinki 8th January 2007

Master of Science Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty Faculty of Science	Laitos — Institution — Department Department of Computer Science

Tekijä — Författare — Author
Ville-Pekka Komulainen

Työn nimi — Arbetets titel — Title
Public Web Services for Ontology Library Systems

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level M.Sc. Thesis	Aika — Datum — Month and year 8th January 2007	Sivumäärä — Sidoantal — Number of pages 60 pages + 18 appendix pages

Tiivistelmä — Referat — Abstract

Ontologies are the backbone of the semantic web vision and ontology re-use is focal for this vision to realize. To promote re-use, public web services are proposed to the ontology library systems. These services can enhance understanding of ontologies by visualizing them, and by offering interfaces for ontology-aware applications to use.

To achieve an understanding of end-user requirements for ontology-based services, high-level needs of different users groups were identified by describing use case scenarios of users concerned in ontology development and utilization. A literature review was carried out covering the support of state-of-the-art ontology library systems, ontology browsers, and ontology interfaces for these tasks. Finally, the proposed services were evaluated by implementing a prototype system Onki covering the requirements. The prototype was tested by integrating it to a web-based semantic annotation system to gain experiences from the service in practice.

As a result, three central user groups involved in ontology life-cycle were identified, which all should be considered individually when developing ontology services. Ontology developers, content annotators and ontology end-users all need specific services, because their viewpoint and especially knowledge about ontologies is different. There is a need for ontology visualization and different levels of programmatically accessible interfaces, ranging from graphical components to Web Services.

ACM Computing Classification System (CCS):
H.3.3 [Information Storage and Retrieval] : Information Search and Retrieval
H.3.5 [Information Storage and Retrieval] : Online Information Services
H 5.2 [Information Interfaces and Presentation] : User Interfaces

Avainsanat — Nyckelord — Keywords
Semantic Web, Ontologies, Ontology Library Systems, Web Services

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpula Science Library, serial number C-2007

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Ontologies [Gru93] are an integral part of Tim Berners-Lee's and others vision of the semantic web [BLHL01]. The vision is an extension of the current www, which enables semantically richer presentation of physical or abstract resources in a machine interpretable and standard format. A key for achieving this is use of ontologies, which describe the used concepts of a conceptual model and their relations in a formal way [Gru93]. On the semantic web, concepts and their relations are identified with a URI and ontologies are usually serialized using standard formats, especially XML, RDF and OWL specified by the World Wide Web Consortium (W3C)[1].

With shared ontologies it is possible to create more accurate information searching methods and also to design intelligent agent-based applications working on a common vocabulary [BLHL01]. However, the semantic web is yet to be established outside the semantic web research community. Aside the fact that only a few ontologies are in real use, one reason for the semantic web waiting to emerge is that, there are only few tools for managing, sharing and using ontologies.

To develop and share ontologies in a systematic manner, Fensel and Ding [DF01] propose ontology library systems as a solution for the problem. These systems should offer tools for managing, versioning, visualization and re-use of ontologies. Knowing that publication and re-using ontologies is the key for realizing the semantic web vision, tools for ontology library systems are needed. This is promoted by the idea of offering public web services for both humans and applications in need of ontological information.

Deploying services into ontology library systems can be beneficial for many user groups in various use cases. First, XML-serialized ontologies are inconvenient for humans to understand and therefore browser-based visualization can help users to understand the ontology's semantic relations and structure as a whole. Second, machine accessible interfaces can be used in external annotation systems to search and use concepts from ontologies stored in the ontology library system. With such a mechanism, the annotation system can be designed as independent from the used ontologies, because ontology managerial and retrieval tasks are left on the responsibility of the server deploying ontologies. Third, programmatically accessible interfaces can be used in semantic search engines and portals to enrich users' search queries by for example using concept-based query expansion methods[JKN01].

[1]http://www.w3.org/

1.1 Semantic Web

Content in the current www is marked up using HTML, a content description language mostly concerned with the presentation layer of the document. Therefore it lacks the needed semantic information about the document, which would enable more intelligent information processing [BLHL01]. For example, when a user queries a web-based search engine using the keyword `java`, the results consist of pages containing the term `java` in any context. The search engine does not and can not understand if the user means the island of Java in Indonesia, the Java programming language or perhaps the coffee-flavor called Java. This is one problem that the semantic web tries to overcome by offering a set of standards for presenting information in a machine interpretable form. By using explicit identifiers for each concept and tagging the content with these identifiers, the search engine could prompt the user to specify the used search term's meaning, thus guiding the user towards the documents he or she was actually looking for. On the semantic web, ontologies are the mechanism for describing concepts and their relations to other concepts in an explicit and machine interpretable format.

To realize the semantic web vision, technological standards are needed for expressing the semantics of resources. W3C has proposed an architectural model and standards for different abstraction layers [KM02] depicted in figure 1. On the bottom of the model, Uniform Resource Identifiers (URI) [BL05] are used for identifying either abstract or physical resources, and Unicode is used to support different character sets. On the next level, XML[2] and XML schema[3] are used as markup language for platform independent information exchange. Next, Resource Description Framework (RDF) and RDF Schema (RDFS) provide essential tools for describing resources and their relations to other resources. This layer enables the building of controlled vocabularies, where resources can have semantically meaningful relations. On the next layer, ontology vocabulary provides mechanisms for creating highly expressive semantical statements about resources. The Web Ontology Language (OWL) is the latest recommendation for this layer. Logic layer makes it possible to create rules for the resources and relations between them. Finally, the proof layer together with the trust layer evaluate whether an application should trust the provided content or not.

RDF is a language for describing abstract or physical resources and metadata about

[2]http://www.w3.org/XML/
[3]http://www.w3.org/XML/Schema

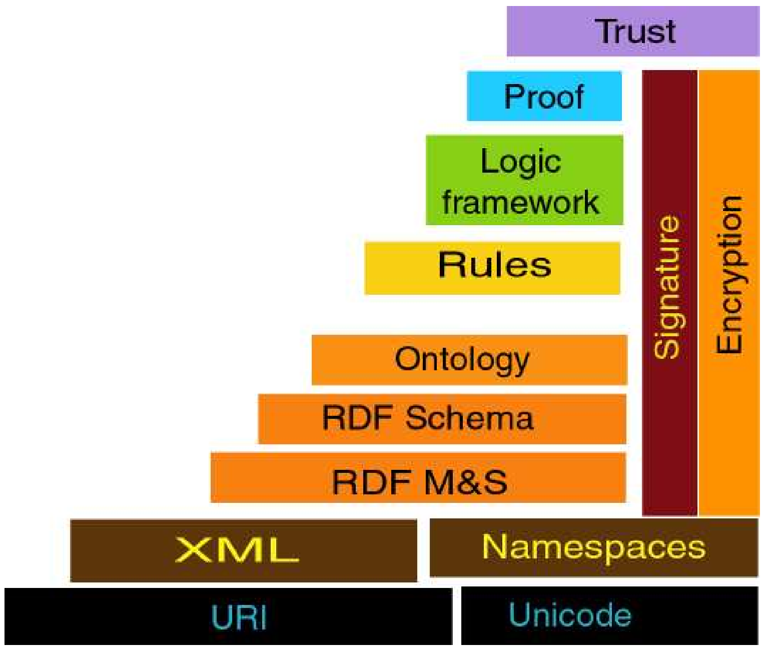


Figure 1: Semantic Web stack by Tim Berners-Lee [KM02].

them [MM04]. It is based on using URI's as identifiers for resources. For example, the URI `http://www.example.org/#CS-Dept` could represent the Department of Computer Science at the University of Helsinki. The basic idea of RDF is to describe statements about resources in a machine processable format. A statement consists of three parts: subject, predicate and object. Subject identifies the resource and object is the value for the subject's property defined as predicate. The object can be an another resource or a simple literal value such as a string or an integer. For example, subject `http://www.example.org/#Helsinki` could have the property `http://www.w3.org/2000/01/rdf-schema#label` having a literal value `Helsinki` for Finnish and `Helsingfors` for Swedish. This graph is illustrated in figure 2.

RDF-graphs can be serialized into machine parseable data using different serializations. RDF/XML and RDF/XML-ABBREV are XML-based formats, which are inconvenient for a human user to read, because there is much overhead from XML tags. The graph in figure 2 could be serialized using RDF/XML as illustrated in figure 3.

Other serialization formats that are supported by popular semantic web tools such

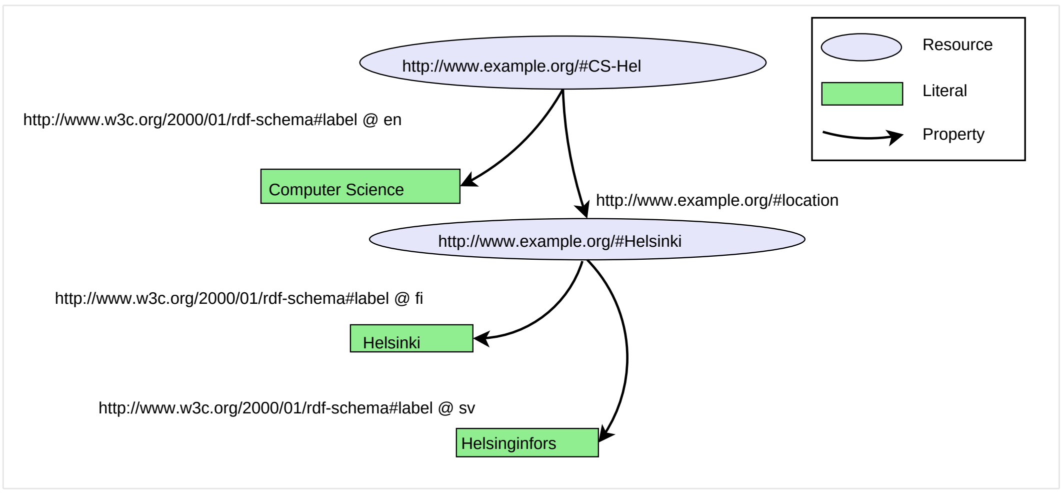

http://www.example.org/#CS-Hel
http://www.w3c.org/2000/01/rdf-schema#label @ en
Computer Science
http://www.example.org/#location
http://www.example.org/#Helsinki
http://www.w3c.org/2000/01/rdf-schema#label @ fi
Helsinki
http://www.w3c.org/2000/01/rdf-schema#label @ sv
Helsinginfors
Resource
Literal
Property

Figure 2: A simple RDF-graph.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
              xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
              xmlns="http://www.example.org/">
  <rdf:Description rdf:about="#CS-Hel" location="#Helsinki">
     <rdfs:label xml:lang="en">Computer Science</rdfs:label>
  </rdf:Description>
  <rdf:Description rdf:about="#Helsinki">
    <rdfs:label xml:lang="fi">Helsinki</rdfs:label>
    <rdfs:label xml:lang="se">Helsingfors</rdfs:label>
  </rdf:Description>
</rdf:RDF>
```

Figure 3: Simple RDF-graph serialized using RDF/XML.

as Jena[4], Sesame[5] and the Protege ontology editor[6] are N-TRIPLE[7], N3[8] and TURTLE[9]. These formats are more suitable for human users than the XML-based formats.

RDF Schema specification defines the classes and properties that one can use to define own classes and properties [BG05]. It defines five core classes for expressing the type of the class, which can be used to differentiate classes from properties and literal values. Nine properties are defined to provide vocabulary developers a stan-

[4]http://jena.sourceforge.net
[5]http://www.openrdf.org/
[6]http://protege.stanford.edu/
[7]http://www.w3.org/2001/sw/RDFCore/ntriples/
[8]http://www.w3.org/DesignIssues/Notation3.html
[9]http://www.dajobe.org/2004/01/turtle/

dard mechanism for describing semantic relations between resources. For example, `rdfs:subClassOf` and `rdfs:subPropertyOf` can be used to express generalization of classes and properties. These relations have similarities with the semantics of inheritance in object-oriented programming. Every instance of Class B that extends Class A is also an instance of Class A. Vocabulary for describing collections, containers, reification and other utilities is also defined. RDF Schema is not intended to cover all the possible definitions that can be used to define classes and properties. For example, it is not possible to specify the minimum and maximum number of values for each property of a class, or to perform set theoretic operations on classes to define new classes. These features and much more are introduced in the Web Ontology Language (OWL).

OWL is an extension of RDFS for users who want to use more expressive semantics in their ontologies [BvHH+04]. For example, OWL defines a more fine-grained set of properties such as `owl:TransitiveProperty`, `owl:SymmetricProperty` compared to RDF's `rdf:property`. In OWL it is also possible to express the cardinality of properties and create restrictions on properties.

There are three variants of OWL to provide different levels of expressivity and reasoning support. OWL Lite extends the RDFS by adding a subset of OWL constructors, but has also some limitations, such as the cardinality of properties can be set to either 0 or 1. The idea behind OWL Lite limitations is to offer a less complex language to build ontologies when compared to OWL DL. OWL DL is intended to support the description logic approach in providing desired computational properties for reasoners. It constraints on the use of RDFS and requires certain constraints to be declared. For example it requires to declare disjointness between classes. The most expressive OWL version, OWL Full, supports all the language constructs of OWL and RDF, but does not share the same computational characteristics for reasoning as OWL DL.

1.2 Ontology Library Systems

Depending on the source, different terms are used when large-scale ontology management tools are discussed. Basically, all the terms refer to server systems facilitating ontology editing, versioning, management and other essential tasks in the ontology life-cycle. For example the terms Semantic Web Management System [VOSM03], Ontology Software Environment [OVSM04], Ontology Server [FFR97] and Ontology Library System [DF01] have been used. For consistency, the term Ontology Library

Systems by Ding and Fensel [DF01] is used through out the thesis.

Ding and Fensel propose ontology library systems as a solution for the lack and difficulty of re-using ontologies [DF01]. In their survey, they suggest that at least the following features should be implemented to facilitate ontology re-use which are also depicted in figure 4. First, management of ontologies should provide open storage, identification and versioning. Second, adaptation of ontologies should be supported by offering tools for browsing and searching concepts and ontologies. Third, a mechanism for editing and reasoning the ontologies should be available and finally, the ontologies should be stored in a standard format. This means, that ontologies should be serialized in a widely used ontology languages such as RDFS or OWL. However, the requirements presented by Ding and Fensel are mostly concerning ontology development phase and the consideration about features needed in using the ontologies from external applications is open. Also, the presented requirements about ontology visualization are on an abstract level.

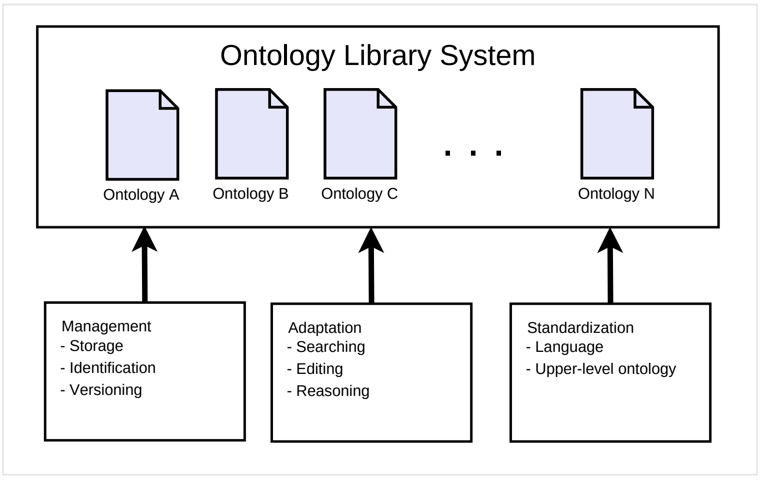


Figure 4: Requirements for ontology library systems [DF01].

Das et al. [DWM01] have studied ontology management system's requirements for business needs in VerticalNet's e-commerce and B2B applications, and they found nine essential requirements. One of them was that systems should have XML interfaces supporting interoperability and ontology sharing . This confirms the need for machine processable interfaces, but there are many protocols and architectural styles available from which to choose.

1.3 Opportunities of the Semantic Web

Although ontologies require much human-intensive work when creating them, once produced they can enhance applications in several ways [McG01]. The adaptations of the semantic web are in enhancing search capabilities of applications, providing common vocabularies for facilitating enterprise information integration, and enabling intelligent agent applications to work with different services in dynamic web environment.

Guinness et al. [McG01] present many examples of ontologies as tools for achieving higher recall and precision in search-engines. Generalization or specialization of search concepts can be used in situations where query returns too few or too many results. For example, if a search for concert tickets in a booking system was done with the term `concert`, the system might respond with a long list of any kind of concerts from classical to jazz. If the system was using an ontology for categorizing different concert types, it could specialize the search concept and suggest the user a more detailed list of concert types to look for. Developing such category-based search-engines could be also established by using traditional database technologies, but the benefit of using ontologies comes from the vocabulary standards such as RDFS and OWL published by W3C, thus making it easier to build generic tools and methods for information processing. Once methods for an ontology are created, it does not matter if the instances populating the ontology are changed — the reasoner for the ontology is still viable.

According to Fensel [Fen01] enterprise information integration could benefit from ontologies, because ontologies provide an extendable platform with explicit definitions of the domain objects. If different systems would use the same ontology to define the processes and data entities transmitted between the systems, then the task of integration would become easier, because fever conversions and mappings between systems would be needed. However, just using ontologies is not enough. The interfaces to systems should be described as WSDL [CCMW01], published as SOAP [BEK$^+$00] and discovered from registries using UDDI[10] for the integration process to be extendable and re-usable. With the use of standard and re-usable components, the workflow between different systems could be straightforward. It remains to be seen if the business processes and domain models of diverse enterprise's are similar enough for achieving collaboration on the basis of a shared ontology.

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

In Berners-Lee's vision [BLHL01] the semantic web's full potential will realize when people start to create software agents searching information from different web sources and exchanging this information with other software agents. Such agents become efficient when more machine processable information exists in the web. For example, software agent designed to find an optimal combination of flight, accommodation and car rental for a person's vacation trip can not negotiate seamlessly with service providers if the services are not described, published and discoverable from a service registry. Moreover, if the services and the data entities are not described using a shared ontology language, the software agent can not understand which services to look for.

These opportunities can not be realized without proper offering and re-use of ontologies. Creating and maintaining them on restricted access slows the emergence of semantic web applications to emerge, and the risk of semantic web becoming an archipelago of individual islands grows [BLHL01]. Ontologies need to be publicly accessible from the public web and system's responsible for ontology management should facilitate human conception of ontologies as well as provide machine accessible interfaces [DWM01].

1.4 Problem Statement

Different approaches for ontology services has been proposed in studies covering ontology library systems [DF01], ontology tools [VOSM03] and visualization of ontologies [FSvH03]. Also, in the field of library science, thesaurus protocols have been studied for accessing online thesaurus [BT04]. However, these studies offer solutions for each specific problem regardless of the context. Profound requirements and an implementation of ontology library systems as public web services are yet to be resolved.

This thesis is bounded to two view-points. First, a human viewpoint for visualizing the ontology, especially in the context of semantic annotation. Second, a machine viewpoint for connecting to an ontology library from external applications. The objective of this thesis is to resolve the services which promote ontology re-use considering these two viewpoints.

Following method was used for approaching the presented problem. Use case scenarios of common interactions between different user groups and ontology library systems were described to identify user requirements at a high abstraction level.

After this, a review at existing solutions was carried out, emphasizing their support for identified user groups and their focal tasks. The presented requirements were derived from the domain of the semantic web, identified use cases and existing solutions. In order to evaluate feasibility of the presented requirements, a system covering the requirements was implemented and evaluated.

The remainder of the thesis is structured as follows. Chapter 2 focuses in motivating and describing the research problem. After this, chapter 3 gives an overview of the state of the art systems proposed as solutions for the presented problem. In chapter 4, the requirements of public web services for ontology library systems are presented, and the system implementing these requirements is described in chapter 5. Finally, in chapter 6 the conclusions of this thesis as well as ideas for future work are presented.

2 Demand for Ontology Services

This chapter describes the need for public web services in ontology library systems by identifying focal user groups and their interaction needs. Services for human users and machine interpretable interfaces are discussed by providing motivating use case scenarios. First, a content workflow example is presented for identifying basic interactions between users and an ontology library system, and different user groups in ontology life-cycle states are identified. After this, the need for visualization is examined in detail and then use cases for semantic annotation are discussed. Finally, programmatically accessible interfaces and web services are considered from the viewpoint of the found user groups.

2.1 Content Workflow Example

To enhance the re-usability of semantic web content, i.e. ontologies and annotations, they can be stored in an ontology library system which is responsible for storing, versioning, editing and publishing of ontologies [DF01]. Different re-use scenarios are illustrated in figure 5 from the viewpoint of ontology library system's public web services. First, the user creates and edits an ontology with a suitable editor and publishes the ontology in an ontology library system for himself and others to use. Here, an interface for uploading the ontologies to the ontology library system is needed. A mechanism for authenticating users is also required to avoid abuse. In the second phase, user creates instances from the supplied ontology, or just simply finds concepts for his annotation, thus enriching his content to the possibilities that semantic web technologies offer. In this case, the user needs a browser for searching proper concepts from the repository and an interface for connecting to the system from a legacy content management system. Finally, the semantically annotated content such as a library's collections are published in a portal for searching and browsing the collections. An ontology-aware portal could use ontology library system's query interfaces for generating view-based user interfaces or exploit semantic query expansion methods for serving the users' needs.

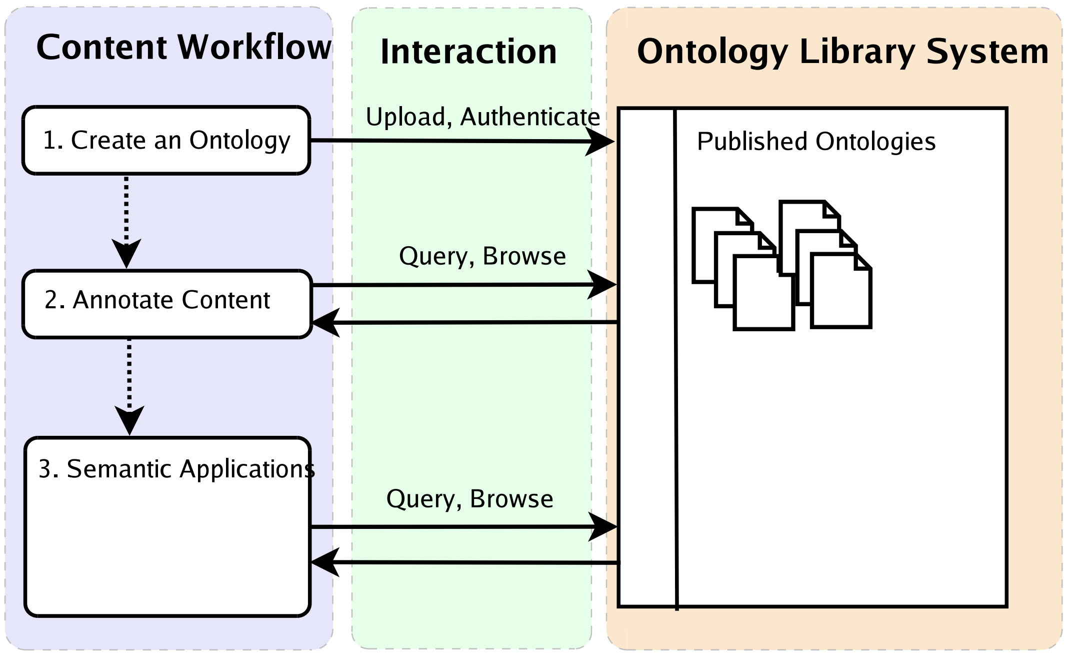


Figure 5: A service utilization example.

2.2 User Groups for Semantic Web Ontologies

Semantic web ontologies can be used to solve different problems in a variety of use cases, such as in information indexing, in information retrieval and in modeling application domains. Three focal user groups can be extracted from the presented content workflow example: ontology developers, users annotating content and users taking advantage of ontologies in their work. Each use case has specific groups involved in ontology life-cycle and all groups should be considered individually in providing ontology services.

Ontology Developers

The first group, ontology developers, are responsible for editing the ontologies and are usually experts in the modeled application domain. They use ontology editors to model the ontology and require all the possible information of the ontology and application domain in their work. Besides this, tools for analyzing and visualizing the ontology as a whole are beneficial for them.

2.2.1 Content Annotators

The second group, content annotators, participate in the semantic web development by creating instances that are linked to available ontologies according to a specific annotation schema or by creating instances of the available classes in ontologies.

Information Searchers and Ontology End Users

The third group consists of users who can be considered as semantic web end-users — both humans and intelligent software agents. They are using ontologies and annotations created by ontology developers and content annotators. For example, users who are developing applications based on ontological data need visualization support for gaining an understanding of the used ontologies as well as machine accessible interfaces for using ontologies that are up to date in the ontology library system.

2.3 Use Cases and Technologies

To understand different user groups' demand for ontology services, use cases and underlying technologies are introduced in this chapter. First, by identifying use cases for browser-based visualization. After this, semantic annotation and arising requirements are discussed. Finally the use cases where machine accessible interfaces in ontology library systems are essential will be presented.

2.3.1 Browser-based Ontology Visualization

Ontologies are usually serialized in a format that is difficult for humans to understand. Therefore browser-based visualization of ontologies is needed to support human understanding of ontologies [PE03]. To realize the requirements for visualization, Fluit et al. have identified three stages in ontology life-cycle where visualization is required [FSvH03]. First, in the development phase, visualization of concepts and their relations is needed for achieving better understanding of the ontology as a whole. As ontologies evolve with more concepts and relations, the more elegant visualization methods are required. Second, when instantiating classes from the ontology using manual or semi-automatic annotation tools, the quality of annotations can be elevated by offering visualization methods, which differentiate instances from classes. Third, after publishing ontologies on the web, tools for analyzing, querying and navigating ontologies are needed.

Ontology developers require additional information supporting the editing process, because ontologies can have numerous concepts and relations between them. Such information is provided by advanced ontology editors and ontology library systems. For example, a method for versioning of ontologies and the possibility of tracking changes between versions is presented by Kauppinen and Hyvönen [KH06]. Seeing the version information and changes between different versions can be beneficial for the ontology developers in their work.

Users instantiating ontologies and annotating content require a clear view to the concepts and their relations for finding the suitable concepts to use. These users are probably not interested in seeing managerial information, such as versions and changes between versions. Their view to ontology might be less technical and their interests could include seeing broader semantic environment of each concepts in a single view. Broader semantic view can guide the users to use more specific concepts, thus elevating the quality of annotations.

Users dealing with ontological information in their daily work. These users might want to see the most trivial and non-technical view to the ontology, because one can not expect them to understand all the underlying semantics such as ontology developers do. As the introduced user group consisting of ontology annotators, also this group's interest does not include to see complex ontology managerial information.

Ontology editors and ontology library systems provide visualization utilities supporting the needs of ontology developers. However, majority of such systems offer only visualization in the editor's user interface which is often a desktop application, therefore requiring additional software to browse the ontology. For example, Protege's OntoViz Tab[11] offers visualization in the editor. The problem in such systems is that they do not provide straightforward publication process to a public www-server, which can have own components responsible for the visualization. The need for independent visualization components is realized and applications managing this has been developed. For example, a cluster map approach for visualization has been suggested by Fluit et al. [FSvH03]. However, implementation of a novel visualization using data structures such as cluster maps or hyperbolic tree [LRP95] often relies on technologies such as Sun's Java Applets[12] or Macromedia's Flash[13] which can not be used without installing plug-ins to the client browser. With increasing deployment of different end devices capable of browsing the internet such as mobile phones, personal digital assistants (PDA) and digital televisions, the need for building applications supporting all platforms and devices is needed. Therefore, common plug-ins in many desktop environments can not be expected to be installed on all clients. To support as many platforms as possible, components responsible for the visualization should be capable of presenting the data in markup languages standardized by W3C such as XML or XHTML[14].

2.3.2 Semantic Annotation

Semantic annotation is a metadata creation process, which aims to enable new information processing methods and extend existing ones [KPO+05]. For example, a webpage of an university researcher typically contains information such as researcher's name, address of the university, research interests and a list of publications. On the current www this information is wrapped inside HTML-tags providing the presentation layout and style of the document, regardless of the actual semantics of the data. On the semantic web, the resources are annotated by describing the relation of the content to other concepts using RDF. If the researcher's web page was described using RDF-descriptions, then the information of researcher's university could reference an instance of a organization ontology's `University` class, and research interests could point to instances of a science ontology's `ResearchField` class.

[11] http://protege.stanford.edu/plugins/ontoviz/ontoviz.htm
[12] http://java.sun.com/applets/
[13] http://www.macromedia.com/software/flash/basic/
[14] http://www.w3.org/TR/xhtml1/

With explicit and formal annotations, it is possible to take advantage of semantic web-based search technologies and integrate information to a research portal efficiently. These annotations can be produced by using different approaches, depending on the desired level of automation and quality of the process. Basically, the semantic annotation process could be classified into three categories:

1. Manual annotation
2. Automatic annotation
3. Semi-automatic annotation

In manual annotation, metadata is described by human users. This is a slow and therefore expensive method, but the quality of the annotations are usually high. Quality means that human user knows the context of the annotation, understands the right disambiguation for terms which can have multiple meanings, and recognizes the meaningful information to annotate. Tools have been produced to support manual annotation of different documents, such as Cream [HS02] for web pages and more general purpose utilities, such as the Protege ontology editor [NSD+01].

Automatic annotation process depends solely on the algorithm responsible for the process. It is based on different information extraction methods, which try to guess the best alternatives for each annotation. Although automatic annotations are more likely less accurate than the ones created manually, Kiryakov et al. claim that automated process provides the needed scalability, and without such automation the semantic web remains only a promising vision [KPO+05]. However, the automation is a complex task and solutions are often domain and even case specific. For example, multimedia content such as images, videos and audio are completely different scenario compared to textual documents, such as web pages. Text-based algorithms can not be applied to binary formats and resolving the context of the annotation is challenging task compared to annotation of textual documents. Fully automated annotation could be used to large document repositories, where achieving high-quality annotations is not as important as the speed, and the data represents single application domain. For example, news articles covering a certain domain such as economics could be feasible to annotate using automated methods.

Semi-automatic annotation begins by applying the algorithms used in automated annotation to the data and then leaving the human user the decision wheater to accept the annotation or not [EMSS00], or the tool may provide for example, the user suggestions for annotations by highlighting parts of the data, and offering suitable mappings to concepts in an ontology. At this point, it is the responsibility of the user to either reject the suggestion, to accept it or perhaps change it some other annotation if the tool offers such functions. Semi-automatic approach combines the benefits of user involvement in manual process and scalability from the automatic annotation.

Annotations, wheater manual or automated need concepts and properties from ontologies. Thus it is essential that systems managing ontologies have suitable interfaces for annotation software to exploit. From the viewpoint of manual annotation, feasible browsing and searching components are needed, and for the automated annotation, the need for machine accessible interfaces is evident.

2.3.3 Interfacing Ontologies Using Web Services

This section covers the needs of identified user groups presented in chapter 2.1 from the perspective of machine interpretable interfaces. In this context, the interfaces are defined as ones which offer an external application the possibility to interact with the provided application. Implementation of interfaces can be based on various techniques: language based API's, communication over HTTP by passing SOAP-messages, REST-architectural approach introduced by Fielding [Fie00], or novel AJAX-technologies [Pau05][15].

A Web Service is defined by W3C as a software system that is designed to support interoperable machine-to-machine interaction over network [IJ04]. Also the interfaces to the services should be in machine interpretable format. To achieve this, W3C enforces the use of standards such as Web Service Description Language (WSDL)[CCMW01] and Simple Object Access Protocol (SOAP) [BEK$^+$00] for describing and publishing the services. When the services are described by WSDL descriptors and published as SOAP interfaces, applications built on different platforms can consume the services as long as they support XML and can communicate over the network.

WSDL is an XML description of the available operations, used data types and endpoints where the services are accessible [CCMW01]. WSDL description contains all the necessary information needed to use the services, which means that WSDL-aware

[15]http://adaptivepath.com/publications/essays/archives/000385.php Idea was first presented by Adaptive Path's J.Garrett

applications can dynamically execute services described using WSDL. A simple example of a WSDL description is a service used for purchasing books. The method for purchasing a book could have the book's ISBN and purchaser identifier as parameters, and it could return the delivery information to the invoking client. The method is mapped to a SOAP endpoint and the data structures are described using XML Schema datatypes. The use of XML Schema datatypes is focal, since this enables language independent description of the value objects, which are transferred between client and server applications. Sophisticated web service frameworks such as Apache Axis[16] have tools for converting WSDL descriptions to language specific value objects and methods. These tools have also utilities that generate WSDL descriptions of language specific interfaces. This lowers the entry barrier of implementing client application invoking remote web services as well as publishing server components as web services.

SOAP is a lightweight, platform independent protocol intended for decentralized messaging with the use of XML messages [BEK$^+$00]. Basically, SOAP is an abstraction layer which binds the underlying language specific interface implementation to a language independent XML interface. Preferred communication layer in many web services frameworks, including Apache Axis, is HTTP[17], but W3C does not restrict using other layers. For example Apache Axis also supports Simple Mail Transfer Protocol[18] (SMTP) and Java Messaging System[19] (JMS).

W3C has founded Web APIs Working Group[20] to gather ideas concerning web-based application collaboration and integration. In the future there might be more interaction between ordinary web applications, because they are resembling more their stand-alone desktop counterparts, and the efforts of the W3C in providing standards for web application interoperability. Although these rising web technologies are yet to gain full industry support, possibilities of using them in interfaces is also discussed in this thesis.

Ontology developers edit ontologies with some GUI-based editor, for example Protege or Swoop[21]. From ontology managerial perspective, these systems provide operations such as loading ontologies from a source, saving the ontology to some repository and management of different ontology versions and their dependencies.

[16]http://ws.apache.org/axis/
[17]http://www.ietf.org/rfc/rfc2616.txt
[18]http://www.ietf.org/rfc/rfc0821.txt
[19]http://java.sun.com/products/jms/
[20]http://www.w3.org/2006/webapi/
[21]http://www.mindswap.org/2004/SWOOP/

Such systems usually have built-in mechanisms for most of these operations, if not for all of them. These managerial services could be provided by the ontology library system, assuming that the services were well thought, covered the necessary functions and were implemented using standard and platform independent techniques. Because ontology developers require these operations for their editing environment, these requirements should be considered when designing an ontology library system. This architectural model would separate the ontology management from the editing work, leaving the suitable tasks for the specialized systems. With this kind of Service Oriented Architecture (SOA) [PL03], the design of ontology editing software is guided towards component-oriented approach, thus making the whole solution less dependent on a single service and the architecture is also less monolithic.

Manual ontology annotation is often done with computer aided tools. In these systems, the ontology is loaded to the application and one can make instances of the defined concepts within the application. This is suitable approach if the users are working strictly on ontologies. However, current state of the art content management systems (CMS) and enterprise resource planning systems (ERP) usually are based on a relational database model. Therefore using specialized ontology-based annotation tools is not always the best choice, because this forces the users to use at least two different systems for the same purpose. Deploying a new information system to an organization encounters often change resistance among personnel. One solution is to update the legacy systems to support semantic web URI's in the data indexing, but this means that software vendors should develop components which can process ontological information and provide presentation layer for visualizing the ontology.

An alternative solution would be to integrate legacy systems to an ontology library system which provides necessary ontology services. First, by providing programmatically accessible API offering all the required query and update services. Second, by providing visualization of the ontology, which could be re-used by external applications. If HTML was used in the visualization, it could be used by stand-alone applications as well as web applications.

Software developers designing tools for the semantic web could benefit from centralized services of an ontology library system. For example, a semi-automatic annotation tool developer probably needs functions for disambiguating term's meanings matching multiple concepts or methods for traversing an ontology's hierarchical structure. Knowing that ontology traversal and concept disambiguation methods are common in ontology processing, they could be implemented to the ontology library systems's interface. Once these services are deployed, software developers can concentrate the interesting problems they are facing. For example, the semi-automatic annotation tool developer can focus in natural language processing.

A user who is searching information from a semantic portal, can benefit from more intelligent search-operations than traditional keyword-based searches. For the end-users to gain full benefit, the portal developer has to implement ontology processing methods to the application. However, this can be a time consuming process and performing complex queries to a large ontology can take away processor time from the actual portal application, if the processing is not on a dedicated server. These services could by provided by the ontology library system, thus allowing the portal developer to concentrate on portal specific issues and leaving the ontology processing to the ontology library system.

An interesting scenario would be to offer rich AJAX-based interfaces, which could be used in generating novel GUI's to the portal. For example, when the user writes a search term to a search field, the web application could query a list of matching concepts from the ontology library using AJAX. The server would then send back a HTML-formatted representation of proper search concepts, from which the user could select the desired search concept. This could guide and help the end-user in the searching task. This idea of using the filter-paradigm familiar from stand-alone GUI's has been implemented in Google suggest[22] and Lyricsfly[23] web-applications, and the idea was introduced into a semantic level in [HM06]. This kind of integration is not the most elegant, because Javascript is not as standardized as other programming languages or Web Services. However, it is easy and fast to implement, and enables re-use of once implemented HTML user interface. Also the efforts of the W3C Web API's Working Group on standardization of different co-operation ways between web applications encourages the idea of lightweight web-based integration.

3 Systems Providing Ontology Services

In this chapter, existing solutions are described and evaluated to realize the need for further development of ontology library system services. First, general ontology development environments and especially the architectural support for interfacing with such systems are evaluated. After this, various systems providing annotation

[22]http://www.google.com/webhp?complete=1&hl=en

[23]http://lyricsfly.com/

interfaces and visualization of ontologies are presente. Finally, a peek towards thesaurus protocols is done to achieve an understanding of the work done in the related library science field. None of the covered systems are adequate just by themselves as a solution to the service need in ontology library systems, but they all offer elegant ideas for solving individual problems. Therefore all systems are evaluated from the viewpoint they were designed for, and how they solve the individual problem of visualization or programmatic access.

3.1 Ontology Development Environments

Ontology development environments focus on the editing and management of ontologies. Depending on the solution, some offer just the tools for editing ontologies and serializing them to a file, while others provide sophisticated features such as programmatically accessible interfaces, ontology visualization, versioning of ontologies, collaboration and user management. Two solutions, KAON [VOSM03] and WebOde [VCFLGP03], are evaluated from the perspective of ontology services.

KAON is a semantic web infrastructure server developed at the University of Karlsruhe. The server is based on a platform which provides a plug-in architecture to develop different modules for ontology management tasks. It provides tools for connecting to the server using a local connection or remotely using Java RMI or Web Services. However, the system uses a proprietary knowledge model extending RDFS, which makes it difficult to use it from external applications, because the semantics is not shared with others and it extends existing standards. The user interface to the system is implemented as web pages with Java Applets, and it offers utilities for browsing and editing classes, properties and instances. The plug-in architecture for different ontology related modules is feasible from technical perspective, but as a whole the framework is fairly heavy from the end-users viewpoint, who needs just the tools to share his ontologies over the Internet. Implementation of KAON's successor, KAON2, is initialized and the main difference between these versions is the support of OWL-DL in the latter compared to the proprietary extension to RDFS in the first version.

WebOde is a system for development and management of ontologies, which also provides middleware components and services facilitating the building of ontology-based applications. As a demonstration of the framework's maturity, it has been used as a technological solution in the Esperonto[24] semantic web portal project. In order to access ontologies from the provided editor or from external clients, the Ontology Access API of WebOde is implemented in Java RMI[25], and these services are also wrapped into Web Services. For visualizing ontologies, a component which creates a browsable catalog from the ontologies is also provided, but no online browser is provided. Although WebOde does not offer end-users sophisticated browsing support or a convenient set of ontology querying services for accessing the ontologies, the framework has strength in well-thought architecture and it offers ontology services such as merging, mapping and evolution which should be considered in an ontology library system.

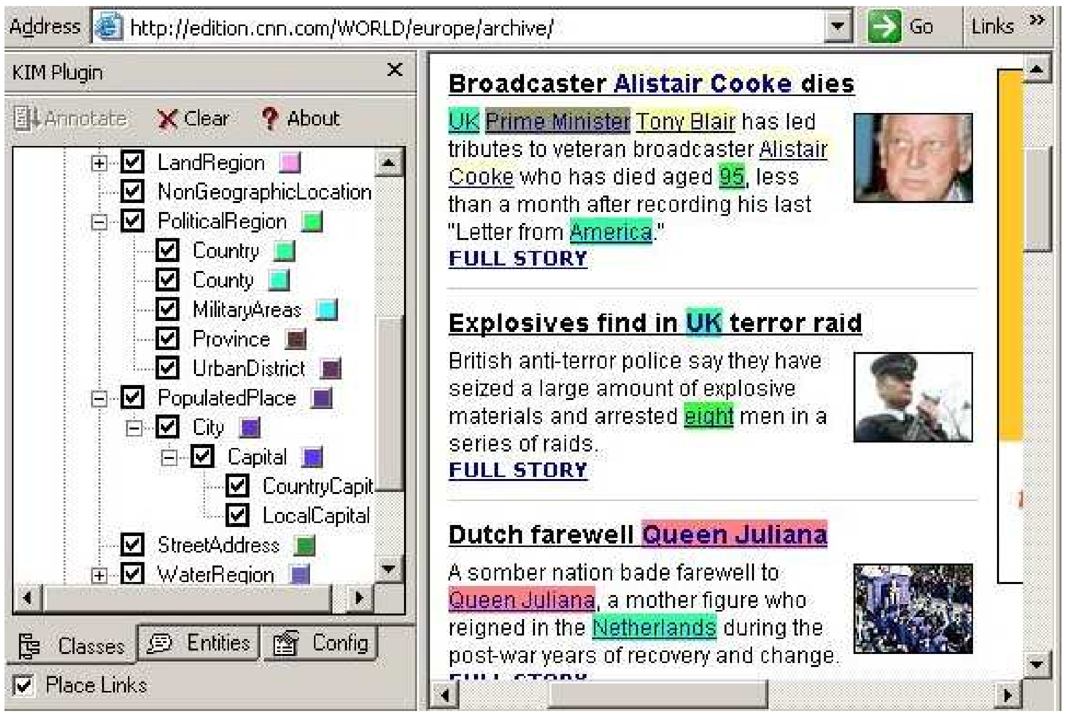
3.2 Ontology Browsers and Interfaces

To enhance re-use and sharing of ontologies, browsers for visualization and interfaces for accessing ontologies have been proposed. Tools range from simple web fronts to novel ontology access platforms used, for example, in automated annotation. In the following, visualization is evaluated from the viewpoint of how these systems illustrate ontological structures to end-users who are not familiar with ontologies. Systems providing interfaces to ontologies are evaluated from a more technical viewpoint. For example, support for SPARQL [PS05] or other semantic web query language should be considered in the frameworks.

The Knowledge and Information Management (KIM) platform was designed especially for semantic annotation, but also as an infrastructure for semantic indexing and retrieval [KPO+05]. Key components in the system are the Semantic Annotation API, Document API, Query & Index API and Repository API. This makes it possible to use KIM just for semantic annotation, or alternatively as an integrated solution for information retrieval which is based on semantically annotated content. For manual annotation, a browser plug-in compatible with Microsoft Internet Explorer is provided as well as a web front for exploring the knowledge base. The browser plug-in illustrated in figured 6 supports semi-automatic annotation of documents by high-lighting suggestions for annotations in the document. After the annotations are approved by the user, one can upload annotations to the annotation server within the plug-in. The platform has sophisticated support for semantic annotation and information retrieval, considering both human users and applica-

[24]http://www.esperonto.net/semanticportal/jsp/frames.jsp
[25]http://java.sun.com/products/rmi-iiop/

tion developers, but the emphasis on annotation reflects the solution as a whole. Although the interfaces cover all the necessary functions, design decisions on using platform specific interfaces such as Java and plug-in support for Microsoft Internet Explorer restrict some users on interfacing with this otherwise skillful solution. However, application developers could prefer such interfaces to generic ones, because for example the Internet Explorer plug-in can be integrated to other applications with ease. Therefore such interfaces can be the ones adopted and utilized more easily by the developer community compared to low-level APIs.



Broadcaster Alistair Cooke dies

UK Prime Minister Tony Blair has led tributes to veteran broadcaster Alistair Cooke who has died aged 95, less than a month after recording his last "Letter from America."

FULL STORY

Explosives find in UK terror raid

British anti-terror police say they have seized a large amount of explosive materials and arrested eight men in a series of raids.

FULL STORY

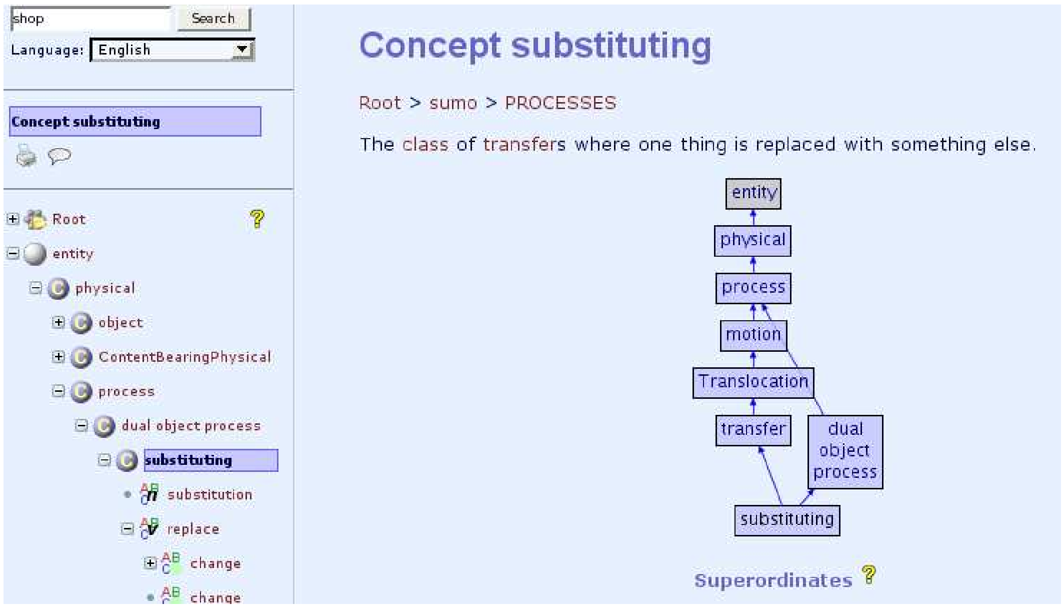
Dutch farewell Queen Juliana

A somber nation bade farewell to Queen Juliana, a mother figure who reigned in the Netherlands during the post-war years of recovery and change.

Figure 6: KIM-browser screenshot.

KSMSA Ontology Browser [Sev03] depicted in figure 7 was designed for browsing Suggested Upper Merged Ontology (SUMO). One goal for the project was to offer a user-friendly view to the ontology for users not familiar with the philosophy of SUMO. The browser offers also an API for accessing SUMO via a C++-library which is a wrapper used in the browser for SUMO-specific ontology querying. However, the provided library does not support remote access over network.

The browser supports features common in many ontology visualization components. For example, class hierarchy of concepts can be navigated by opening branches from the left-hand side and the description of the class is provided at the same time on the right-hand side of the screen. The classes can also be searched by a keyword search. A unique feature in the browser is the support for multi-lingual textual representation of SUMO-axioms in natural language. The browser offers a decent visualization of the underlying ontology, but the solution is case-specific, which explains partially the novel features such as textual representation of axioms and image-maps of the class hierarchy. However, these features are worthwhile to be considered in ontology visualization, because users who are not familiar with ontologies might understand complex axioms and class hierarchies more clearly using images and textual representation of axioms.


Concept substituting

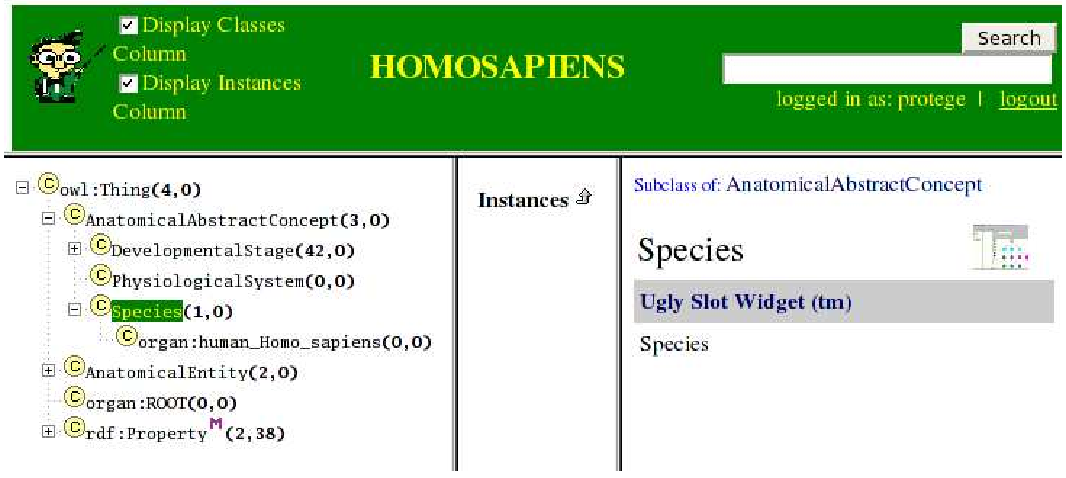
Root > sumo > PROCESSES

The class of transfers where one thing is replaced with something else.

Figure 7: SUMO-browser screenshot.

The Protege community has developed a web-interface[26] to the popular Protege ontology editor for sharing and browsing ontologies over internet. The interface illustrated in figure 8 provides a similar view to the knowledge base as the stand-alone application. It is also possible to make annotations from the client, but ontology editing is not possible. Adding an independent component to Protege for visualizing the ontologies for web-users makes is possible to separate ontology development from re-use and browsing needs of different user groups who are not interested in ontology development. However, the requirements of application developers is not considered in the form of machine accessible interfaces. At the moment developers need to use Protege's Java-API, which can not be used from remote clients.

[26] http://protege.stanford.edu/plugins/protegebrowser/



owl:Thing(4,0)
AnatomicalAbstractConcept(3,0)
DevelopmentalStage(42,0)
PhysiologicalSystem(0,0)
Species(1,0)
organ:human_Homo_sapiens(0,0)
AnatomicalEntity(2,0)
organ:ROOT(0,0)
rdf:Property(2,38)

Subclass of: AnatomicalAbstractConcept

Species

Species

Figure 8: Protege Web Browse screenshot.

SchemaWeb[27] is a simple portal for storing and querying ontologies. The portal serves human users by supporting the following features: browsing the available schemas, search them by keywords, query all the schemas simultaneously using an online form, and tools to submit ontologies to the repository. For software developers and agents, there are interfaces for querying the schemas using either SOAP or REST. At the moment there are more than 200 schemas stored in the repository. Although SchemaWeb is basically just a directory for storing ontologies having very little other functionalities, the concept might be something the end-users of ontologies are interested in. The idea of having an easy-to understand web front where users can search for ontologies, submit them to repository and query them with proper machine accessible interfaces should be taken into consideration as an user interface of an ontology library system. The approach of keeping functions as simple as possible might not be rejected by the end users compared to feature-rich, but complicated front-ends. Unfortunately, software running the portal is not publicly available.

3.3 Programmatic Access to Controlled Vocabularies

Thesaurus protocols provide interfaces for navigating controlled vocabularies by semantically meaningful relations. Similarities in the programmatic access to these systems is considered, because thesauri are controlled vocabularies such as ontolo-

[27]http://www.schemaweb.info/

gies and they are used in creating metadata about resources. Thesauri often have at least the following relations defined between the terms: related term, narrower term, broader term and preferred term [ABG04]. These can be described easily by using semantic web technologies. For example, all the presented relations can be described by using the Simple Knowledge Organization System (SKOS)-format introduced by W3C[28]. Binding [BT04] et al. have studied thesaurus protocols in their work and recommend basic ideas for enhancing the protocols. For example, a single query to a thesaurus should always return all the related terms of the queried term. From a technical viewpoint, they suggest that services should cache the results of the queries to provided needed performance. These ideas are also applicable to ontology service in some extent.

SKOS API[29] is an interface for accessing thesauri described in the SKOS-format. SKOS is a W3C proposal for describing thesaurus in OWL. Basically it maps many of the common relations used in thesaurusi by utilizing OWL Full semantics. The motivation is to get users developing thesauruses to use a standardized format, which is interoperable with semantic web languages. Using the standardized SKOS-format makes it possible to use the same reasoner for all thesauri described in SKOS. The lack of standardized interchange format and access to thesauri has restricted a wider use of thesauri, which often contain semantically meaningful information.

The SKOS API is based on previous work on thesaurus access [BT04], which provided a basic set of operations for querying a thesaurus. The API has several useful functions, which can be used to query the underlying thesaurus and is implemented using Java. Furthermore, the provided Java-interface is exposed as Web Services. Suggested operations for thesaurus access – which are quite self-explanatory – are listed in table 1. For example, calling the method `getConceptByPreferredLabel` with parameter 'car' the concept corresponding the label is be returned. After this, it is possible to retrieve all the relatives of this concept by calling the method `getgetConceptRelatives` with the concept as parameter. Binding and Tudhope created a simple pilot browser to gain experiences of the API [BT05]. They learned that providing a fixed set of operations is not always adequate and semantic web query languages such as SPARQL should be supported.

The suggested API offers a starting point for standardized thesaurus interaction from the viewpoint of an application developer in need of thesaurus access. However, it

[28]http://www.w3.org/TR/2005/WD-swbp-skos-core-spec-20051102/

[29]http://www.w3.org/2001/sw/Europe/reports/thes/api/docs/

Method Signature
`getAllConceptRelatives(Concept c)`
`getAllConceptRelativesByThesaurus(Concept c, URI thes)`
`getAllConceptsByPath(Concept c, Relation r, int dist)`
`getConcept(URI Uri)`
`getConceptByExternalID(String externalID, URI thes)`
`getConceptByPreferredLabel(String preferredLabel, URI thes)`
`getConceptRelatives(Concept c, Relation r)`
`getConceptRelativesByPath(Concept c, Relation r, URI thes, int dist)`
`getConceptRelativesByThesaurus(Concept c, Relation r, URI thes)`
`getConceptsMatchingKeyword(String key)`
`getConceptsMatchingKeywordByThesaurus(String key, URI thes)`
`getConceptsMatchingRegex(String regexp)`
`getConceptsMatchingRegexByThesaurus(String regexp, URI thes)`
`getSupportedSemanticRelations()`
`getSupportedSemanticRelationsByThesaurus(URI thes)`
`getTopConcepts(Concept c, URI thes)`
`getTopmostConcepts(URI thes)`

Table 1: SKOS API methods in interface SKOSThesaurus [DB04]

can not be applied straight-forwardly to generic ontologies, because the assumption on fixed set of used properties which exist in SKOS can not be done. Also, the lack of methods which can be used to store information into the thesaurus could prove useful, and as suggested by Binding and Tudhope, support for semantic web query languages should be considered. Latest version of the API Javadoc[30] intimates that support for semantic web query languages has been considered.

3.4 Suggested Approach for Ontology Services

Various solutions for ontology re-use, utilization and visualization have been developed, each concentrating in either one of the areas, or providing a scalable all-purpose workbench managing all of them. Large scale solutions provide most of the necessary functions required in ontology library systems public web services, but trade-off for this is often the lack of end-user involvement in the design phase, mak-

[30]http://www.w3.org/2001/sw/Europe/reports/thes/api/docs/

ing such systems difficult to use. As in contrast, lightweight solutions concentrating in a specific problem such as novel visualization are not as scalable or lack important requirements from other perspectives.

Ontology visualization is an important factor in ontology re-use for users who are not experts in ontological engineering, because it is the first user interface that ontology users will confront when searching an ontology in an ontology library system. One ontology repository providing user friendly portal is the SchemaWeb RDFS-repository[31], where users can find all the ontology related information without unnecessary technical details.

A layout for the browser, where concepts are visualized on the left side of the screen and properties on the right, is favored in many solutions such as KIM, Protege Browser and KSMSA Ontology Browser. A hyperbolic tree visualization [LRP95] is feasible for small ontologies, but the graph might get tangled on large ontologies consisting of numerous classes and properties. Other useful features found in most of the solutions are: searching classes and instances by keyword, textual representation of axioms, localization and partial graphical rendering of ontologies using images. However, none of the presented solutions considered different user groups in the visualization.

Each of the presented solutions have solid ideas to enrich from ontology re-use and utilization approach. KIM and SKOS API suggest a group of services that should be included as an integral part of ontology library systems public web services. Methods for navigation the ontology and the support for semantic concept based expansion should be offered, because these are easier for the developers to adopt. However, a standard query language interface should also be deployed [BT04]. Issue that has not been tackled, is the return format of the web services methods. If SKOS Web Services API would be used, then it would be easy to deserialize the XML into language specific datatypes, because the returned objects are described in XML Schema datatypes, which can be deserialized for example into JavaBeans[32]. However, this is not possible for general ontologies because the SKOS approach works on fixed set of classes and properties, instead of working on a general OWL model where it is not feasible to enumerate all the possible classes in an ontology. As suggested in SKOS API, metadata about the stored ontologies should also be accessible programmatically. For example, information about used properties in

[31]http://www.schemaweb.info

[32]java.sun.com/products/javabeans/

the underlying ontologies and the human readable labels of the concepts is useful. Technically, the web services standards should be favored as an alternative for a language specific ones, because ontologies are described in machine independent format, therefore the service for them should also be. As discussed in chapter 2, the emergence of AJAX technology and work of the W3C WebAPI working group makes it possible to create re-usable user interface components to ontology library system for web applications to re-use.

The range from heavy server solutions managing all aspects of re-using ontologies to simple lightweight ontology visualization is vast, therefore leaving a gap for a solution providing the essential re-use and visualization services from end-users' viewpoint. Such solution should combine strengths from all the presented systems, but still avoid the pitfall of making it too complex. Therefore there is a need for end-user centric approach solving the needs of different user groups in a user-friendly manner.

4 Requirements for Ontology Services

This chapter describes a proposal for the requirements of ontology library system's public web services. These requirements were derived from the use cases presented in chapter 2 and from the evaluation of existing solutions in chapter 3. Requirements are expressed in an abstract level, to avoid verbose requirements specification and considering scope of the study. he classification of requirements is done using established methods.

The requirements were classified into user requirements and system requirements, which is the separation recommended in many software engineering studies and books, such as [Som00]. User requirements reflect end user's need from a non-technical viewpoint, as in contrast the system requirements are presented from a technical perspective. System requirements were categorized into two categories: functional and non-functional as advised in [RO85]. Each system requirement, both functional and non-functional, has a source, which is a user requirement expressing an end-users actual need. Functional requirements model the user's needs about what the system should do and what kind of services it should offer, for example, the possibility to interact with other software through an interface. Non-functional requirements constrain the provided services or functionalities. Such constraints can be performance related, such as maximum response time on a web page request, or constraining to use certain standards or protocols in the implementation. For example, the use of XML in interfaces or extendable system architecture system can be considered as non-functional requirements.

4.1 User Requirements

User requirements of the three user groups involved in ontology life-cycle were extracted from the scenarios described in chapter 2. These requirements are presented as a UML use case diagram [GB05] in figure 9. Any user, either a human or machine, is considered as an actor in the UML use case diagram. There are four actors who need services from the ontology library system: 1) ontology developers, 2) content annotators, 3) information searchers and 4) external applications. First three user groups are human users and the last describes an external software, which is used by the human users. External application can be for example a content management system, ontology editing system or a portal's search-engine. A detailed descriptions of use cases and the corresponding user requirements are presented in the following section. First presenting rationale for each requirement and then the requirement itself.

Download Ontologies

Content annotators, ontology developers and external applications need a service where they can download and reference ontologies to use in their work. Therefore ontology library system should provide interfaces for both applications and human users for downloading different versions of stored ontologies.

Store Ontologies

To encourage users in sharing ontologies, and to provide a service for publishing ontologies as a centralized service, an easy-to-use service for uploading ontologies is required. Ontology editors and external applications should be provided with a service for uploading new ontologies and updating existing ones with new revisions. To avoid abuse of a random user updating ontologies maintained by other users, the service should be accessible only by authenticated users.

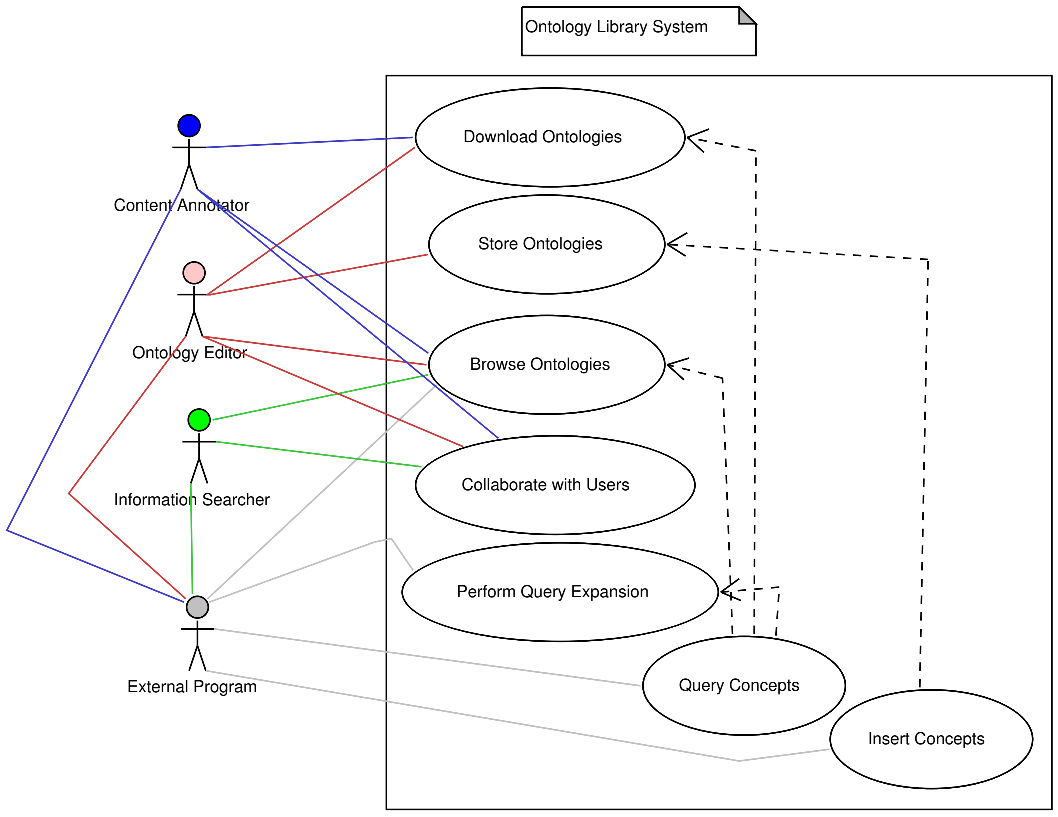


Figure 9: Ontology Library System. An end user's use cases.

Browse Ontologies Online

All actors using an ontology library system require ontology visualization for gaining a clear perspective to the ontology as a whole. The ontology browsing component should provide operations for navigating the ontological relations and searching concepts by keywords. The browser should also provide different abstraction levels to the ontology according to the user group, because users have different knowledge about ontologies. To facilitate the work of developers building ontological applications, visualization components should also be re-usable from external applications.

Collaborate with Other Users

Ontologies can have dependencies with other ontologies. For example a concept in an ontology A can extend a concept in an ontology B. Therefore developers of depending ontologies need to communicate with each other about possible changes that can create conflicts to depending ontologies. The collaboration can also happen between content annotators and ontology developers. To support collaboration between users, a communication system between users should be provided.

Perform Query Expansion

External applications can benefit from services offering query expansion for concepts. With intelligent information retrieval algorithms using query expansion, a higher recall can be achieved for example in search engines. A group of programmatically accessible services should be provided, which enable applications to search concepts by keyword and expand concept definitions by traversing ontological relations. Metadata about stored ontologies, such as root concepts, used properties and properties reflecting human readable labels of concepts should be provided to enable dynamic composition of queries.

Query Concepts

Service for querying concepts is required by external applications to enable interaction between an application and the ontology library system. Therefore a semantic web query language interface with convenience methods should be provided to an ontology library system. Such services can be used internally by the ontology browser interface.

Store Concepts

Users annotating content can benefit from a interface, which provides a service for storing annotations in the ontology library. With such a service, it is possible to share instances of concepts with other users. To achieve this, an interface for uploading annotations to the system should be provided, but the services should be restricted to authenticated users. Otherwise there is a risk that malicious users abuse the service by deleting or modifying annotations.

4.2 Functional Requirements

This chapter describes functional requirements for the end-user services. Such services include: browsing ontologies, searching concepts by a keyword search, providing utilities for semantic annotation of content and interfaces for querying concepts.

Provide Browsing Interface to Ontologies

As suggested by Ding and Fensel [DF01], ontology browsing is a key element in an ontology library systems. Therefore, a browser providing HTML[33]-pages of concepts and their relations should be offered. Essentially, the browser should visualize each concept and related concepts clearly to the user. Because human readable labels of the concepts can be described using different languages, the user interface should also support internationalization.

Different Views to Concepts

Users with different knowledge about ontologies have different requirements for ontology visualization as discussed in chapter 2.3.1. This should be supported in the ontology browser by providing alternative views to concepts depending on the users needs. To achieve this, there should be a switch in the user interface, which changes the properties shown to the user. For example, ontology developer's view should include all the technical details of concepts as in contrast for content annotators and information searchers the view should be less technical and provide as much information in natural language form instead of RDF-triplets. This is especially beneficial for users migrating from thesauri to ontologies in information indexing, because thesauri contain less technical information compared to ontologies.

User Authentication

Functions such as messaging between users and storing ontologies to the system require an user authentication. In most cases, ontology browsing and querying can be done without authentication, but it should be considered that not all ontologies are necessarily public in the future. For example, some intra-enterprise ontologies could contain either delicate user information or the ontology is seen as a source of revenue and organizations do not want to publish it without fee.

Application Programming Interface (API)

Communication between external applications and an ontology library system is not possible without exposing services as interfaces. Especially operations for navigating the ontology by semantic relations and searching concepts by keyword should be

[33]http://www.w3.org/TR/html4/

supported, because such operations enable for example query expansion. Interface presented in SKOS API [DB04] contains most necessary functions, but the services should be altered to support generic ontologies.

A lightweight API offering ready-to-use user interface components would facilitate integration of external applications, because invoking client can use once generated user interface in the ontology library system. This can be implemented either by providing a library containing language specific graphical widgets such as Java Swing, or alternatively the components can be delivered as HTML pages using asynchronous HTTP-request (XMLHttpRequest)[34].

Store Annotations

Users annotating content can benefit from a service for storing annotations to an ontology library. With such a service, users are able to share there annotations with each other. This service requires that there is an interface for storing the instances. It should be considered that a public interface that writes content to an ontology library requires users to be authenticated before any transactions are done.

Store and Retrieve Ontologies

Central requirement for an ontology library system is to offer services for users to store ontologies to the system, as well as the opportunity to retrieve them for further use. It should be considered that ontologies evolve [Sto04] over time with new published versions and therefore the interface for storing and retrieving should manage different versions of one ontology. This implies that there should be an operation for querying a list of all available ontologies in the system, as well as an operation for resolving all the versions of selected ontology.

Messaging Services for Users

To enhance collaboration between users, an integrated messaging system should be designed to the system. For example, an organization A uses an ontology developed by organization B and is interested in sending change requests for the ontology. Change request could be a demand for a new concept to the ontology. Because ontology library system provides browsing services for the ontology, it could also

[34]http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060927/

offer a feedback channel for ontology developers. This could be implemented as a simple feedback-form in the browser.

4.3 Non-Functional Requirements

Non-functional requirements describe the requirements from a technical viewpoint and have effect on the design and implementation of software. Applications managing and providing services for ontologies have non-functional characteristics which are described in the following.

Scalability

Semantic web ontologies can be very small covering a well-defined domain ontology or large constructed of existing thesauri having numerous classes and instances such as the General Finnish ontology [HVK$^+$05]. Also, an ontology library can contain many ontologies, and multiple clients can access the system simultaneously. Therefore, the system should scale well even with large ontologies and a heavy load.

Web Services Interfaces

Interfaces of an ontology library system should be accessible from all clients developed on different platforms, because ontologies are also described in machine independent format. Therefore, a proper protocol for the interfaces is SOAP with corresponding WSDL descriptors, because they are modeled using XML and are not platform specific.

Configurability

Ontologies can be described using different classes, instances and properties and also be serialized using various formats. For example, different properties can be used to describe hierarchical relations of concepts or human readable labels. Furthermore, ontologies can be stored either in a file, database, revision control system or on a remote HTTP-server. Therefore it is essential that a system implementing ontology services is highly configurable. Without adequate system configuration, the services can only be used with certain kind of ontologies.

Support Standard Ontology Languages

As suggested in [DF01], any ontology library system should support standard ontology languages recommended by W3C, especially RDFS and OWL. However, the application logic should not depend on the underlying ontology language and connection to ontologies should be decoupled from the application logic.

Extendable Architecture

The architecture should be extendable in a sense that the provided services can be modified without changes to the public interfaces. In object oriented programming this means preferring interfaces over classes. For example, a system can define an interface for a general synonym repository, which can be used in searches if an implementation for the interface is developed. A motivating example of extendable architecture is the Protege ontology editor, which has many interfaces for programmers to develop plug-ins such as the OWL-editor.

Layer Architecture

The purpose of the layer architecture is to separate presentation, application logic, data access and data from each other to enable a system design where all layers are independent [BMR+96]. In such an architecture each layer is only coupled to the layer below it and therefore changing a layer, for example the presentation tier, does not effect the system as a whole. Layer architecture is essential for ontology-based applications, because semantic web technologies like RDF-frameworks are under constant evolution. For ontology library system web services, a three-layer model is proposed. On the bottom there is the data access layer which is responsible for retrieving concepts from the ontology storage such as a database or a file. This layer does not have any application logic, it just queries ontologies. After this there is the application logic layer, which queries concepts using the data access layer. This layer is responsible for providing all the application logic, such as expanding the user's queries using external synonym repositories. Finally, there is the presentation layer which consists of HTML pages that are generated by classes that control the user interface flow.

5 Onki — An Implementation of Ontology Services

This chapter describes the architecture and technical details of the implementation for ontology library system's public web services. These are the end-user services for the ontology library system described in [KVH05] and [KH04]. A prototype realizing the requirements presented in chapter 4 was implemented to evaluate the suggested requirements in practice. However, user authentication and messaging services were not implemented in the prototype, because they are not core services for ontologies compared to, for example, visualization. A demonstration of the presented software is available at `http://demo.seco.hut.fi/onki/`.

Onki-system consists of three components, which all can be used by each user group in different use cases. First, the ontology browser is a web application providing ontology visualization. Second, a subset of the browser's visualization components can be used from external applications by the provided Javascript API. This provides re-usable user interface components for web applications. Third, web service interface (SOAP, WSDL) provides access to the underlying ontologies for platform independent interoperability.

Figure 10 illustrates Onki from the end-users viewpoint. In the inner circle, there are the services provided by an ontology library system. Such services are for example ontology storage and versioning. Public web services demonstrated in Onki are on the outer circle providing the end-user services. Such decoupling of core ontology services from end user services enables the services to be implemented independently from each other. Next, the functional components for different user groups are depicted with gray boxes. Content annotators are provided with a web service interface as well as Javascript API, which can be used to create annotations from external web applications. Ontology developers can use the ontology browser to see the contents of the underlying ontologies as a whole. Users who are searching information or are working with ontological information can use the browser to find the right concepts for their use.

5.1 Architectural Overview

This chapter describes the general architecture of the implemented public web services and how the system meets the presented non-functional constraints in chapter 4.3. First, it is described how non-functional requirements are satisfied and after this the architecture is explained using use case diagrams and UML package diagrams.

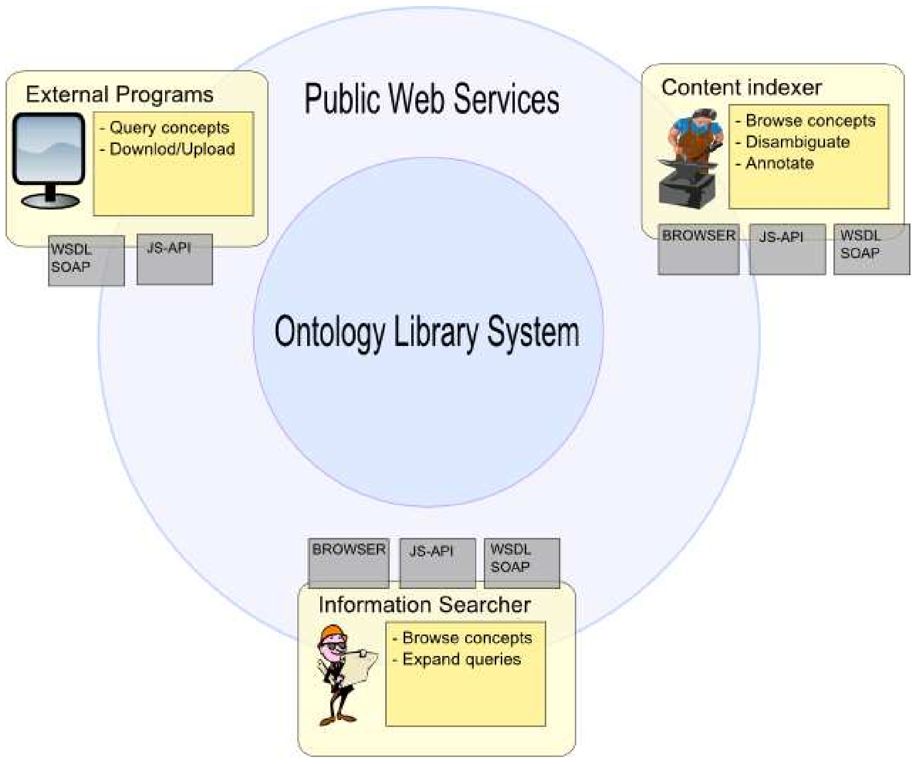


Figure 10: Supporting users in their tasks.

Finally, Onki configuration is unfolded to achieve an understanding of how the system can be adjusted to different ontologies. Scalability was achieved by using Jena semantic web framework, which is capable of storing ontologies in a database. It is also possible to use it as an in-memory RDF-storage, which results in faster queries but trading of memory space. For example an ontology of 1,000,000 RDF-triplets requires approximately 2 Gb of RAM from the server. One of the key benefits of using Jena is, that it supports W3C standardized ontology languages such as RDFS and OWL.

Onki was designed and implemented using a layered architecture [BMR+96], which enables modular design from the user interface to ontologies. With the separation of user interface, application logic and data access, theoretically all layers are independent of the each other. The architecture is depicted in figure 11. At the bottom there is the data layer, which represents the ontologies serialized in a standard format. Ontologies can be physically stored in a local file, remote HTTP endpoint or a database. Second, the data access layer is responsible for querying and storing ontologies to ontology repository. This layer encapsulates connection to ontologies using the Data Access Object (DAO)[35] pattern. This enables that application layer can access the ontologies regardless of the underlying RDF framework. Next, application logic layer offers services which can be used at the user interface layer. These services are also exposed as Web Services, thus fulfilling the requirement of platform independent connectivity. Finally, the presentation layer offers end-users browsing and searching utilities. The presentation layer has also a Javascript API, which enables lightweight integration of other web applications to the ontology library system. The provided Javascript API has functions for remote applications to query concepts easily from the server. Concepts matching queries are returned in an HTML block providing a visualization of the ontology.

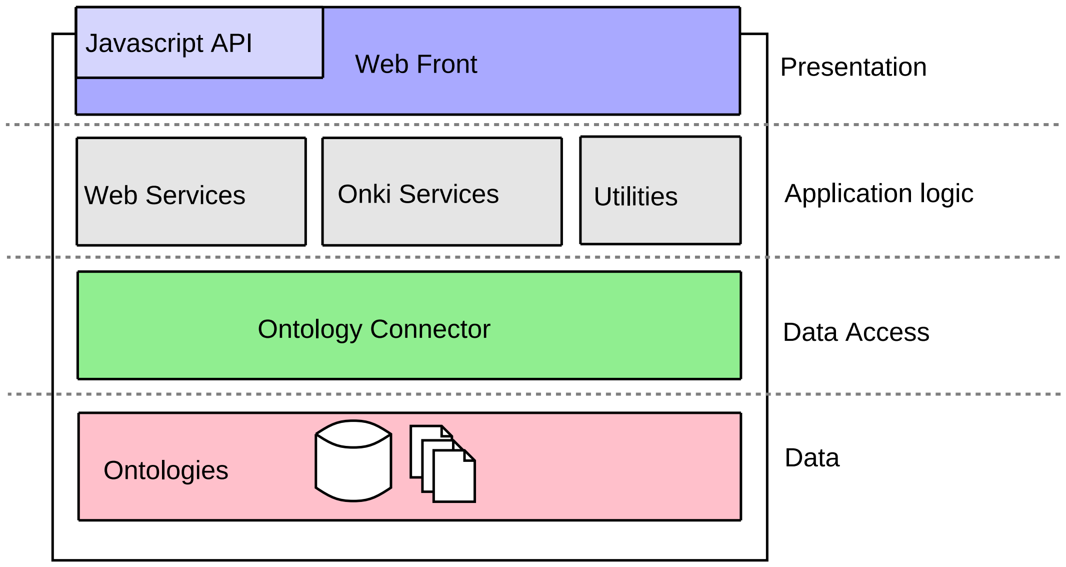


Figure 11: Layer architecture.

From a more detailed design viewpoint, a UML diagram illustrating central package structure is depicted in figure 12. From this figure it is possible to realize the layered architecture, as each package contains classes from a single layer. These packages and core classes in them are described in the following.

- **fi.tkk.seco.onki.dao** This package contains interfaces and classes responsible for ontology access. There are separate implementations for connecting to ontologies stored in either file, database or HTTP address. For example, implementing IOnkiDao interface it would be easy to provide a connection to

[35] http://java.sun.com/blueprints/patterns/DAO.html

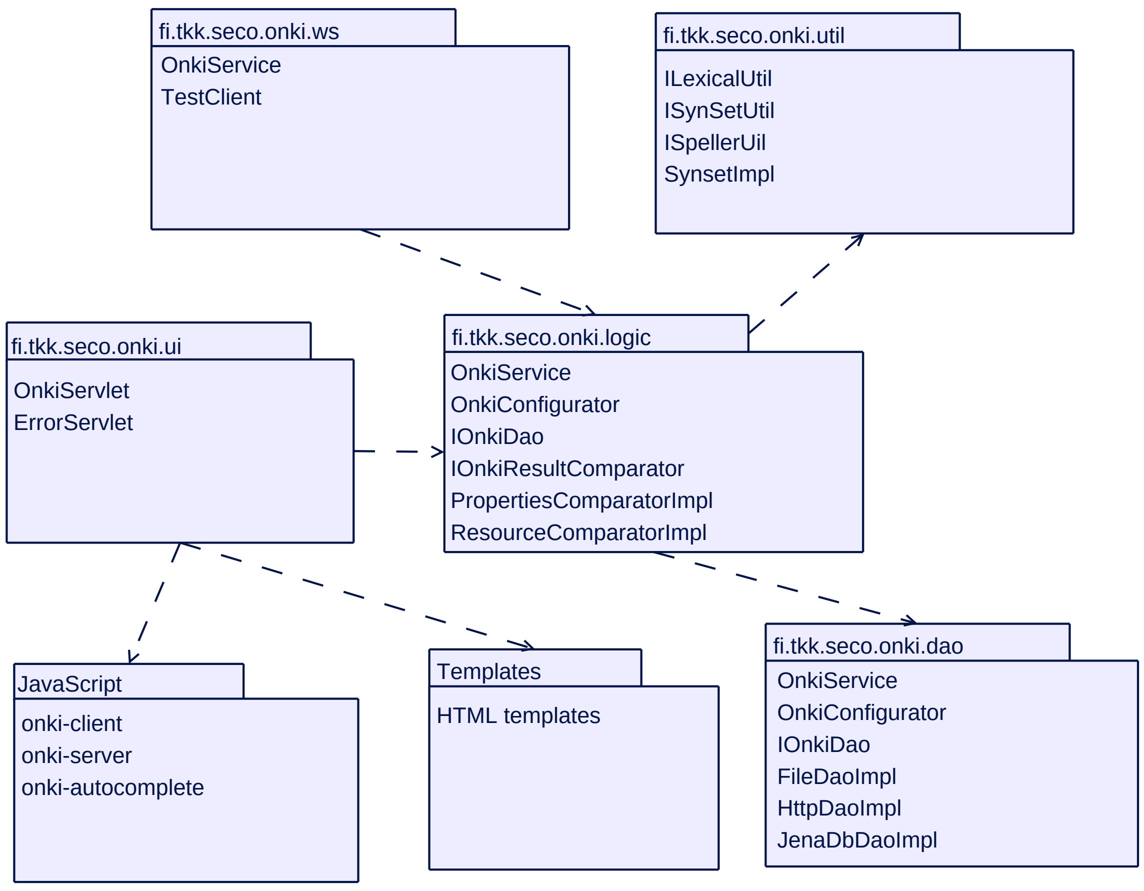


Figure 12: Onki Package diagram.

ontologies stored in a revision control system repository, such as CVS[36].

- **fi.tkk.seco.onki.logic, fi.tkk.seco.onki.util, fi.tkk.seco.onki.ws** Next, there is the application layer, which contains classes for ontology processing. This layer represents Model in the Model-View-Controller (MVC)-design pattern. Application logic is responsible for creating queries to ontologies and also for binding concepts together based on defined properties, such as `rdfs:subClassOf`. Additional logic, such as using external lexicons or synonym libraries is defined and implemented here.

- **fi.tkk.seco.onki.ui, Javascript, templates** Finally, the user interface layer represents View and Controller in the MVC-design pattern. Java Servlets are responsible for controlling application flow and Apache Velocity[37] templates

[36]http://www.nongnu.org/cvs/
[37]http://jakarta.apache.org/velocity/

are used for creating HTML pages.

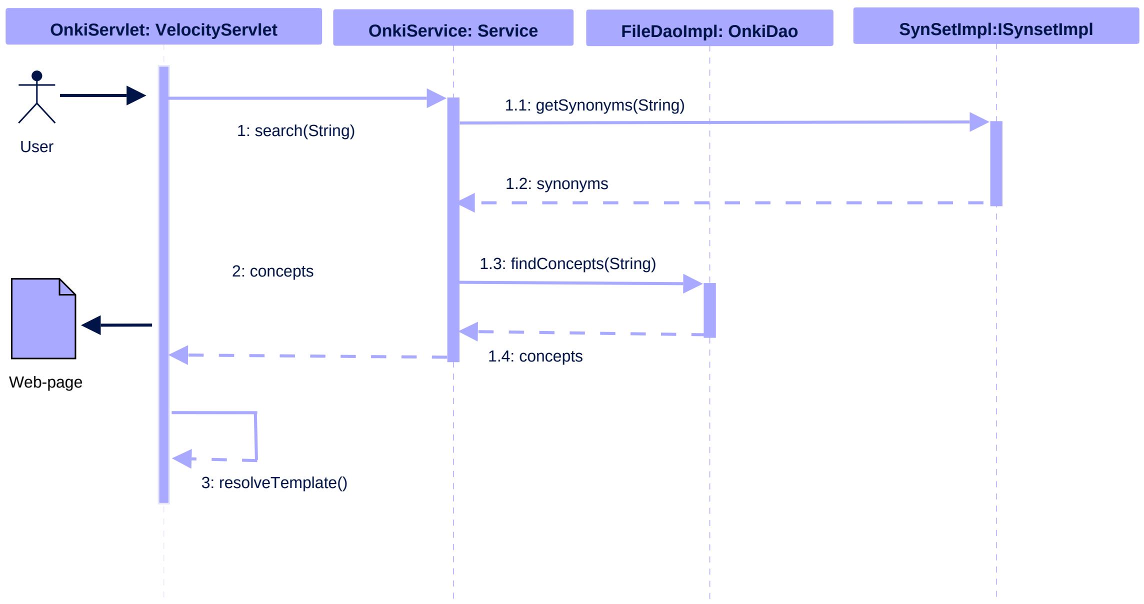


Figure 13: Onki sequence diagram.

For achieving a broader view to the architecture, a UML sequence diagram of a simple service is provided in figure 13. This is a common use case of an ontology library system's web service, where end-user searches for concepts. For example, a user searches for a concept defining a pupil for his annotation. First, user types search string 'pupil' to her browser's search form and the query is processed in `OnkiServlet`, which delegates query to the component responsible for application logic, the `OnkiService`. If an utility for resolving synonyms for the query string is configured, then the service resolves possible synonyms using provided interface `ISynSetImpl`, which offers a synonym repository connection. For example, the synonym-repository could resolve the term 'student' as a synonym of 'pupil'. Now, after the synonyms are resolved, both of the search strings, 'student' and 'pupil' are dispatched for the ontology access layer. Here, Jena is used to query the ontology for concepts whoese labels match either of the search string. If the ontology contained only a concept that had a label 'student', then the user's query 'pupil' had not matched any concepts. With the introduced utility for resolving synonyms from an external repository, the user's query matched the concept with label 'Student'. After this, the flow return to the controller servlet, which resolves the right HTML template for user's request. Finally, the resulting concept with it's hierarchy and semantical relations is shown to the user. This sequence is similar to the use case where query is executed from a remote client using either SOAP or provided Javascript API. Only the invoking client changes and the flow from `OnkiService` follows the same sequence.

Used ontologies and visualization is configured using an XML configuration file, which enables deployment of different ontologies to the server. Configuration has elements for adjusting visualized ontological properties and the structure of the ontology. For example, it is possible to configure the properties used to create class hierarchy and also to specify properties used for text-based searching. Additionally, configuration controls the sorting of query results, localization of the user interface and configuration of external utilities. For brevity, each configuration element is described briefly and thorough configuration with additional comments is described in appendix 1.

Ontology Repositories

The most essential configuration element specifies the used ontologies, because it allows the Onki to be used with various ontologies. In this setting it is possible to configure `1..n` ontologies to the system. Additionally, it can be specified if the used ontologies are located in a local file, relational database or HTTP address.

Annotation Repository

Besides the ontologies for browsing and interfacing via services, also a repository for storing annotations can be configured. Such repository is beneficial for content annotators, because this provides them a possibility to store and share created instances of ontology concepts. The annotation repository is separated from the main ontology repository for two purposes. Fist, it is easier to separate annotations from the ontologies, if this is later seen as a necessary operation. Second, when ontologies and instances are stored in different locations, it is possible to use in-memory storage for ontologies and database for annotations.

Root Concepts, Properties and Property for Creating Hierarchy

To create an hierarchy of the concepts and properties for the visualization, root concepts of ontologies must be specified in the configuration. `1..n` root concepts can be specified for a single ontology. Also, a property for creating concept hierarchy must be configured. Although ontologies are often constructed using the `rdfs:subClassOf` hierarchy, not all share this characteristic. For example, in SKOS-thesauruses hierarchical relations between concepts are developed using `skos:narrower` and `skos:broader` which are not sub-properties of `rdfs:subClassOf`.

Textual Representation of Concepts

The textual representation of concepts is usually described using a specified property, for example `rdfs:label`. In Onki, one can specify `1..n` properties, which are shown to the user when browsing concepts within the system. The properties specified here are also used in searching concepts.

Alphabetical Index of Concepts

An Alphabetical index can be beneficial for users who are not familiar with the hierarchical approach in looking up for concepts. To create an alphabetical index of concepts, all the alphabets from which the indexes are created must be enumerated in the configuration. Alternatively the index could be generated in runtime, but for large and evolving ontologies real-time determination of index alphabets can be time consuming.

Concept Sorting

To sort the concepts and properties of concepts in the user interface, a configuration specifying this is required. Without such sorting, the visualized concepts would be in a random order, because concepts do not have any special property specifying their internal order within an ontology. To achieve sorting of concepts, one can specify a property, which is used in sorting. For example, it can be specified that `rdfs:label` or any other property is used for sorting the results of a user's search.

Localization

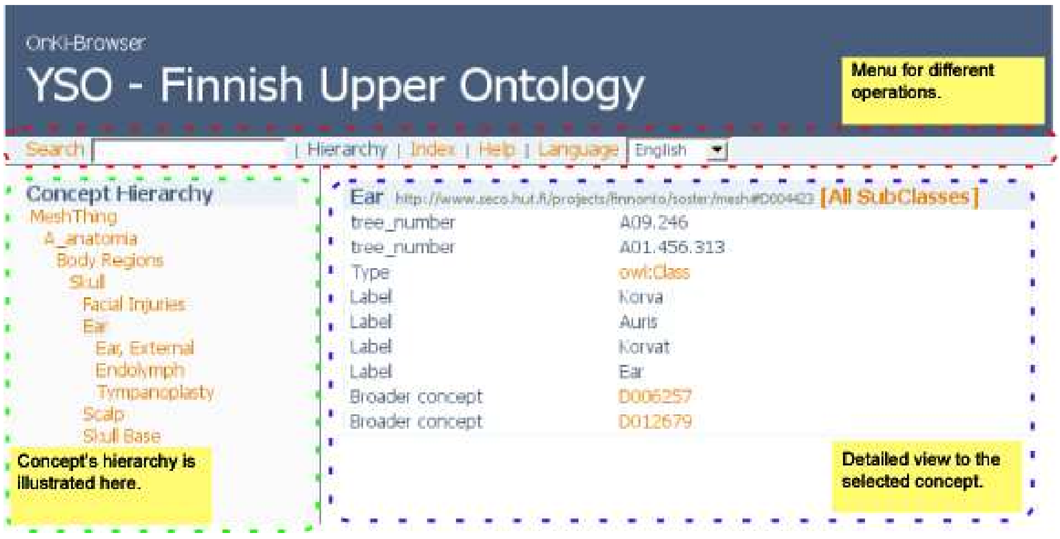
To enable multilingual user interface, there is a parameter for specifying all the possible localizations of the user interface. It is also possible to configure the default language for the user interface. Technically, localizations of user interface are stored in a separate localization file called a 'resourcebundle'.

External Utilities for Extending Queries

Because ontologies do not always contain all the lexical information of the concepts, a utility interface for adding such tools to the system is provided. These are used only when user does keyword queries to ontologies. Onki provides interfaces for this, and also a simple file-based implementation for synonym-repository.

5.2 Onki-Browser – Visualization of Ontologies

Onki-Browser is used for illustrating concepts relations, for searching concepts and for exploring the ontology as a whole. The browser can also be used for annotating content from web applications. A screenshot of the browser visualizing the concept `Ear` in the biomedical MeSh-ontology[SGD04] is depicted in figure 14. Browsing functions are available in the navigation menu identified with a red rectangle. Concepts can be searched using the search field and the ontology's hierarchy can be explored from by navigating concepts relationships illustrated in the green rectangle. The hierarchy contains all the generalizations of the concept, sibling concepts and direct sub-concepts of the currently browsed concept. In the concepts view, all properties related to the concept are listed in the blue rectangle. This is the view that is shown to the user when a concept is selected by either searching a concept using search field, selecting it from the alphabetical index or traversing ontological relations from other concepts. Because concepts can have different textual representations depending on the language, it is possible to change the localization of the user interface from the provided combobox. When the user changes the language, Onki-browser resolves and visualizes corresponding language of the concepts in the user interface.

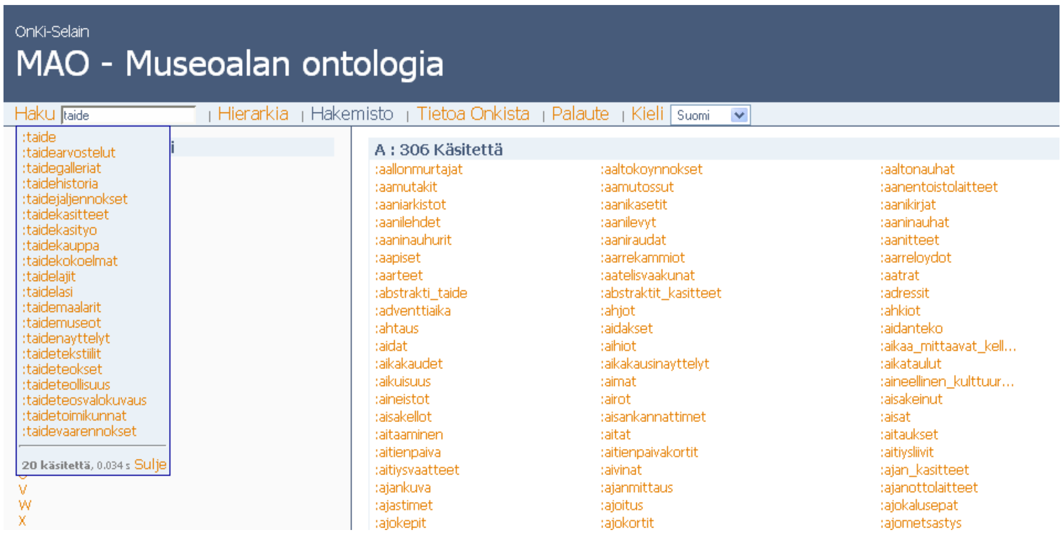


Ear http://www.seco.hut.fi/projects/finnonto/soster/mesh#D004423 [All SubClasses]

tree_number	A09.246
tree_number	A01.456.313
Type	owl:Class
Label	Korva
Label	Auris
Label	Korvat
Label	Ear
Broader concept	D006257
Broader concept	D012679

Figure 14: Basic view to the concepts in the browser.

In addition to traditional browsing between concepts based on their semantical relations, Onki provides a search utility for searching concepts matching the concept's labels to users query. An example of user querying for concepts matching for 'taide' is illustrated in figure 15. When the user enters characters to the query field, application sends the query after every character pressing to the server using XmlHttpRequest. This technology enables asynchronous messaging between Onki's client-side Javascript code and server-side Java-code. Such messaging provides fast response to user's search, because only the query is sent to the server and the results are shown in a lightweight layer on the screen. Additionally, user receives instant feedback to his query to see if there were any matching concepts.



:taide
:taidearvostelut
:taidegalleriat
:taidehistoria
:taidejaljennokset
:taidekasitteet
:taidekasityo
:taidekauppa
:taidekokoelmat
:taidelajit
:taidelasi
:taidemaalarit
:taidemuseot
:taidenayttelyt
:taidetekstiilit
:taideteokset
:taideteollisuus
:taideteosvalokuvaus
:taidetoimikunnat
:taidevaarennokset

20 käsitettä, 0.034 s Sulje

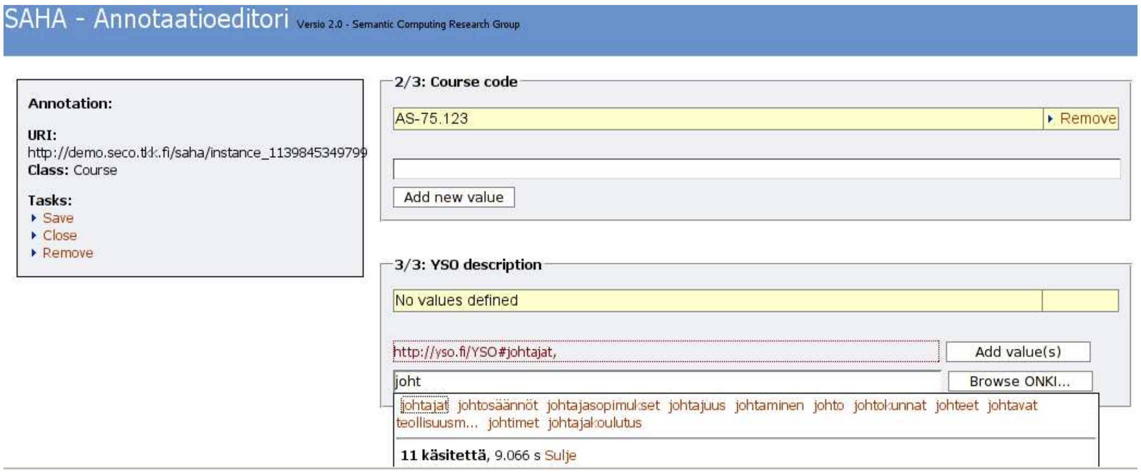
A : 306 Käsitettä

:aallonmurtajat	:aaltokoynnokset	:aaltonauhat
:aamutakit	:aamutossut	:aanentoistolaitteet
:aaniarkistot	:aanikasetit	:aanikirjat
:aanilehdet	:aanilevyt	:aaninauhat
:aaninauhurit	:aaniraudat	:aanitteet
:aapiset	:aarrekammiot	:aarreloydot
:aarteet	:aatelisvaakunat	:aatrat
:abstrakti_taide	:abstraktit_kasitteet	:adressit
:adventtiaika	:ahjot	:ahkiot
:ahtaus	:aidakset	:aidanteko
:aidat	:aihiot	:aikaa_mittaavat_kell...
:aikakaudet	:aikakausinayttelyt	:aikataulut
:aikuisuus	:aimat	:aineellinen_kulttuur...
:aineistot	:airot	:aisakeinut
:aisakellot	:aisankannattimet	:aisat
:aitaaminen	:aitat	:aitaukset
:aitienpaiva	:aitienpaivakortit	:aitiysliivit
:aitiysvaatteet	:aivinat	:ajan_kasitteet
:ajankuva	:ajanmittaus	:ajanottolaitteet
:ajastimet	:ajoitus	:ajokalusepat
:ajokepit	:ajokortit	:ajometsastys

Figure 15: A user querying concepts using autocompletion.

To provide efficient text-based searching of concepts, Onki queries are executed to match all the labels that are configured to represent the concept's human readable labels. Furthermore, an the synonym-repository can be used to expand a single search term to multiple search terms. Using more than one label for searching, and having the possibility to connect to a synonym repository provides higher recall to users searches. However, the precision in such searches can be lower, because the expanded query matches more concepts. A better implementation of the utility would expand the query in such cases when the user's query doesn't match any concepts.

The features for searching and browsing ontologies are useful for illustrating the ontology and gaining an understanding of the ontology. Furthermore, these features can be used from external web application using the provided Javascript API. As a proof-of-concept, the browser was integrated to the content annotation system SAHA [VH06] illustrated in figure 16. First, the user types first characters from the concept that he or she wants to use as an annotation. After this, the query is sent to Onki and the results are rendered to the invoking client browser, and annotation system user can select one or more of the concepts to the application by clicking them. The Javascript API has functions for binding Onki-Browser and remote systems to each other by using few simple API methods.


SAHA - Annotaatioeditori Versio 2.0 - Semantic Computing Research Group

Annotation:

URI: http://demo.seco.tkk.fi/saha/instance_1139845349799
Class: Course

Tasks:
- Save
- Close
- Remove

2/3: Course code

AS-75.123 | Remove

Add new value

3/3: YSO description

No values defined

http://yso.fi/YSO#johtajat, | Add value(s)

joht | Browse ONKI...

johtajat johtosäännöt johtajasopimukset johtajuus johtaminen johto johtokunnat johteet johtavat teollisuusm... johtimet johtajakoulutus

11 käsitettä, 9.066 s Sulje

Figure 16: Using browser's components from external web-applications.

5.3 Onki-API – Programmatic Access to Ontologies

The idea of Onki-API is to offer facilities for querying concepts, but also to store annotations to the system. Provided interface is not intended as an semantic web application framework – such utilities already exist, for example Jena, Kowari [WGA05], Sesame [BKvH02], OntoViews [MHSV04] and Protege. The API consists of a web service interface and a Javascript API, which provide different mechanisms to interact with Onki. Web services provide platform independent integration, as in contrast the Javascript API can only be used from a web application. Using web services interface requires more time for integration compared to the Javascript API. However, the Javascript API is more of an experimental solution compared to standardized web services.

Web Services

SOAP and WSDL was chosen as the implementation, because they provide platform interoperability and a standard language for describing and invoking services. Conventional communication frameworks such as RMI were not considered, because they are restricted to language specific implementations. One possible implementation technique, the REST-method [Fie00] introduced by Fielding, was not considered because the tools are yet to be developed and it is more of an ad-hoc method compared to standardized web services. However, REST is very similar to SOAP – communication is done over network using, for example HTTP and XML is used to transfer data. However, the invoked services can not be parameterized as web services.

SOAP methods were generated by using the Apache Axis-toolkit[38]. The toolkit generates WSDL descriptors and SOAP endpoints from a specified class or interface automatically. The provided services are listed in table 2 and a more detailed description is in appendix 2. These services were implemented, because they provide application developers tools for navigating the ontology's structure, and finding concepts using free-text search.

For example, a hardware store's clerk annotates his product catalog for tools' material descriptions. First, an annotation application could query concepts that match the user's query 'Iron' using the method `getConceptsMatchingLabel('Iron', 'en')`. The method returns an RDF/XML-serialization of the concepts matching the query and the application presents the results to the user. In this case the concept 'Iron' would be returned. If the user is not satisfied with the returned concepts, the application could have a built-in mechanism for searching all the subconcepts for the returned concept. This is achieved by calling the method `getSubConcepts('Iron', true)`. Using this method all the transitive subconcepts would be returned to the user. The ontology could contain, for example, the concept 'Stainless steel' as a subconcept of the concept 'Iron', and it would be returned to the user.

Each service for querying ontologies returns an RDF/XML-serialization of the statements matching the query. These services resemble functions in SKOS API, but the methods have been generalized to support any ontologies, because SKOS API is based on the SKOS knowledge model. The knowledge model in SKOS specifies a fixed set of properties for concepts, and therefore it can not be applied to an ontology library system supporting different kind of ontologies.

However, when the services were tested using a simple demonstration client, it was found that using web services to transfer concepts or subgraphs of ontologies is challenging, because the RDF-model does not fit well to the web services concept. Web services use transfer objects as the parameters of the services. The transfer

[38]http://ws.apache.org/axis/

Method and input parameters, all return a subgraph matching the query
`getConcept(uri)`
`getConceptsMatchingLabel(match, lang)`
`getConceptsMatchingRegex(regexp, lang)`
`getOntologies()`
`getOntologyProperties(uri)`
`getOntologyLabels(uri)`
`getRelatedConcepts(uri, dist)`
`getRootConcepts(uri)`
`getSiblingConcepts(uri)`
`getSubConcepts(uri, transitive)`
`getSubConceptsMatching(match, subClassOf, begin, end, xmlLang)`
`getSuperConcepts(uri, transitive)`
`querySparql(query)`

Table 2: Onki web services API.

objects[39] are described using XML Schema and later transformed to language specific objects in the invoking client. General ontology services can not be specified to use pre-determined format for concepts, because then the services could not be used for all ontologies. The return format of Onki Web Services is a subgraph of the ontology matching the query. This is a generic approach, which can be used for all ontologies, but as a trade-off requires much RDF-processing in the invoking client. This is resolved in SKOS API by using a specific transfer object representing the SKOS-concept (`skos:concept`). However, this approach can not be used in different ontologies, because concepts between ontologies can be different from each other. This fundamental difference between object-oriented software engineering and ontological engineering is challenging for the development of ontology services.

Javascript API

An alternative integration method was implemented to provide application developers an API to use Onki's visualization components from web applications. It was implemented as a Javascript interface, which allows web applications to interact with Onki. The lightweight interface enables use of Onki-visualization by query-

[39]http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html

ing concepts from the repository and binding the found concepts to the external application. The services were packaged into a Javascript library, which can be distributed and re-used by including the library from the used Onki-server. To test the applicability of the library, services were piloted in a web-based annotation system SAHA [VH06].

Two different methods for using the Javascript API were implemented. First, an AJAX-based solution, enabling asynchronous communication with the ontology server. Second, a window referencing approach, which offers the whole user interface to be used from other applications. A technical detailed description of the API is in appendix 3. Both approaches are based on the idea of linking a web application's HTML-form to an API-method, which searches concepts from Onki. After the user finds the desired concept, the URI of the concept can be linked to the calling application's HTML-form. Both approaches offer the ability to either search concepts matching a search string, or alternatively constrain the search to subconcepts of a certain concept. Using a subconcept constrain can be useful, because annotation schemas properties have often `rdfs:range` limiting the set of appropriate concepts. For example, an annotation schema for artwork can define that a value for the property defining the artistic style must be an instance of certain concept from an art-ontology.

The concepts and flow for using the interface is illustrated in figure 17. In the AJAX-approach, the user types in characters of the name of the concept he or she is looking for. After each character entry, the search string is sent to Onki-server, which returns an HTML-layer for the calling application. From this layer, the user can select the desired concept to the calling application by clicking the concept. Technically, each returned concept is attached with a link which binds the clicked concept's URI to the HTML form's input field in the client application. The Onki-browser can be used as a whole by using the window-referencing approach depicted also in figure 17. In the image red rectangles describe user's actions, either selecting a link or clicking a button. This solution is technically similar to the AJAX-approach, the basic idea is to bind HTML-form's input fields to the called API-methods. Using window-referencing, Onki-browser is opened to the user, where he or she can take full advantage of the ontology visualization and provided search mechanisms. After the desired concept is found, the user just selects the concept by clicking the Fetch concept' link in the browser. The URI and label are transferred to the calling application's HTML-form's fields.

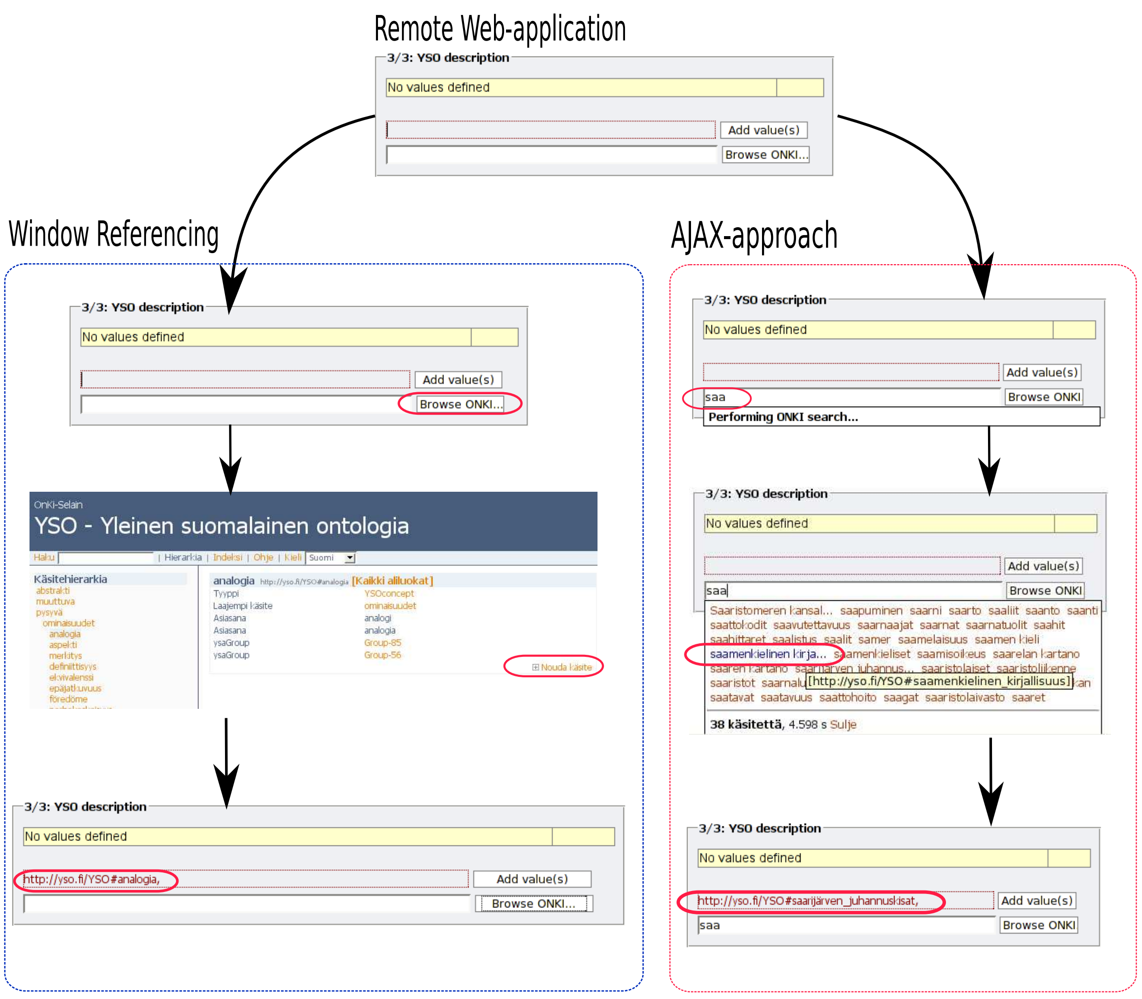

Remote Web-application
Window Referencing
AJAX-approach
3/3: YSO description
No values defined
Add value(s)
Browse ONKI...
saa
Performing ONKI search...
YSO - Yleinen suomalainen ontologia
http://yso.fi/YSO#analogia,
http://yso.fi/YSO#saarijärven_juhannuskisat,
38 käsitettä, 4.598 s Sulje

Figure 17: Two ways to use Onki Javascript API.

However, the provided Javascript API has limitations which are listed in table 3. AJAX-approach can not be used in all circumstances, because current web browsers are only capable of creating XmlHttpRequests to a server which is in the same domain as the client application. For example, a web application in the domain `http://www.helsinki.fi/` can not communicate using XmlHttpRequest to a server in the domain `http://www.tkk.fi/`. This is restricted in all browsers, because there are security issues which might be exploited by hostile users. Fortunately, this restriction can be by-passed in Mozilla Firefox and Microsoft Internet Explorer by adjusting browser settings.

In the Internet Explorer, one can set the security level for each domain so that it is possible to use XmlHttpRequest from different domains. Mozilla Firefox introduces the idea of signed scripts, which can be used to authenticate the request and therefore

Application deployment / Integration method	Window referencing	AJAX-approach
Application and Onki are in the same domain	MS Internet Explorer	All browsers
Application and Onki are in different domains	-	All browsers
Using a by-pass proxy	All browsers	All browsers

Table 3: Browser support for Onki's Javascript API.

remote calls to a different domain is permitted. The restriction can also be by-passed by using a proxy script. The idea is that the remote calls are forwarded to a local address, which forwards the requests internally to the remote server. When such a mechanism is used, the browser has no problem in creating the XmlHttpRequest, because the calls are sent to a local address – although the requests are actually forwarded to the remote server.

The alternative integration using window-referencing has similar security-related restrictions, when the applications are not in the same domain. However, in contrast to the AJAX-approach, the restriction in window-referencing can not be bypassed by any browser settings. The window-referencing can only be used with a bypass proxy if the application and Onki are on different servers. Using Ajax in the communication lowers the need for bandwidth, because only a small part of the page is fetched compared to a whole page. Also the response time is faster than in window-referencing.

6 Discussion

Objective of this thesis was to resolve what services are needed to an ontology library system, and how to implement them. A human and a machine viewpoints were considered with respect to application possibilities in the semantic web domain. This was achieved by first discussing the general service need for ontologies and then evaluating how the state-of-the-art solutions support these objectives. After this, the requirements were presented. A subset of them was implemented in the ontology services prototype, Onki.

This thesis presented the end-users's need, requirements and different approaches for providing public web services to an ontology library system. Suggested services offer end users an interface for browsing and searching concepts in underlying ontologies.

The services also enhance ontology re-use, because ontologies can be published to the server and used from external applications by using the provided programmatically accessible interfaces. Prototype implementation for demonstrating the services was also introduced and evaluated.

The requirements, user profiles and implementation ideas for the services were identified by describing use case scenarios of tasks which are related to ontology re-use. It seems that performing thorough user requirement elicitation by user-centric methods could have been more feasible, but such methods were not used for two reason. First, semantic web ontologies are relatively new and potential end-users are not familiar with the arising possibilities. Second, requirements elicitation is out of the thesis's scope.

Ontology libraries, ontology visualization tools and ontology services were evaluated for achieving an understanding of the shortcomings in state-of-the-art systems. Each solution offered solid features to a certain user need, but all lacked end-user centric approach to the services. Also, the evaluated systems did not consider the concept of offering these features as a centralized service.

6.1 Services for the End-Users

The result of the study were the proposed services for an ontology library system, and the identified user groups of such services. Functional and non-functional requirements as well as the implementation was considered from an end user perspective. Following user groups were found in need of ontology services: ontology developers, content annotators and users in need of ontological information. All user groups were found to have individual requirements for the services as well as overlapping ones. Although different user groups had similar service needs, each user group should still be offered a tailored version of the services to avoid pitfalls in many existing solutions.

It was found that there is a need for services, which provide visualization of ontologies. Also, interfaces for querying concepts and creating annotations to an ontology library system are useful. To be more specific, the visualization and browsing should offer mechanisms for illustrating ontological hierarchies and various search mechanisms, such as keyword search. Also, possible internationalization of ontologies should be considered in the user interface. From the interface point of view, services should offer methods for traversing the ontology's hierarchy and offer tools and methods disambiguating search terms. A semantic web query language interface is also required, since pre-built methods are often inadequate.

A suggestion of non-functional requirements for the services was proposed describing what are the technical constraints and requirements. The system implementation should consider the following technical features: scalability, configurability and extendability. Demand for these arise from the semantic web domain because ontologies can contain numerous concepts and be very different by structure. Providing web services interfaces ensures interoperability on different platforms, because WSDL and SOAP does not address the implementations programming language. Finally, layered architecture ensures that the system is easy to modify, and different user interfaces or services can be developed to the system.

As a proof of concept, a part of the introduced requirements were realized in a prototype implementation, which provided a platform for evaluating the requirements in practice. To support ontology visualization, a web application providing a hierarchical view to the ontology was implemented. It also offers detailed view to concept definitions and utilities for searching. Decision of implementing a standard web application was preferred over techniques offering novel visualization such as Macromedia Flash, because the system had to be re-usable from all clients. To support ontology re-use, especially considering annotation and search engines, web services interfaces for querying and storing annotations to the ontologies was developed. However, using web services requires external application developers to code the user interface to ontologies and therefore an alternative lightweight Javascript API was introduced, which enables fast and easy integration to the system.

6.2 Initial Experiences

Ontology visualization using a web application was considered to be useful, especially as a communication channel between ontology developers, but also between ontology developers and end-users. However, browsing and visualization as sole services were not as essential as the Javascript API built on top of the browser. Need for ontology visualization tools and the concept of public ontology portal will gain importance when ontologies are published and adapted outside the research community.

The idea of lightweight Javascript API as an integration interface for web-applications was adopted in two demonstrations. First, it was tested in a commercial content management system's pilot version supporting semantic annotation. It was found that Javascript API supported rapid application development, because almost all the functionality, including the user interface is in the ontology library system. The second adaptation was the web-based annotation too SAHA [VH06] developed in the Semantic Computing Research Group at the Helsinki University of Technology. The idea of searching concepts for annotation using Javascript API allowed the application developer to focus in the annotation software, rather than to ontology visualization. This ensured that the ontologies are up to date, because it is on the responsibility of the ontology library system. From software performance viewpoint this model also removes the burden of external application from processing large and memory-greedy ontologies.

Unfortunately, the Javascript API can be used only when both the ontology library system and an external application are deployed on the same server. This limitation arises form the risk of cross-site scripting, which can be seen as a security issue. There are technical solutions to avoid this limitation, but it does not remove the fact that the Javascript API is unstable for industrial usage, because Javascript, and especially XMLHttpRequest implementations vary between browsers and the suggested solutions for avoiding the limitations are not standardized. However, W3C has established a Web APIs Working Group[40], which works on standards for web-application interfaces. If the working group offers feasible standards which are implemented in forthcoming browsers, then presented Javascript API can be a noteworthy integration method.

To test the web services in practice, a simple client was developed for demonstrating different query expansion techniques. Available methods were useful for query expansion, because they covered many common tasks such as disambiguation, concept generalization and specialization. However, from a technical viewpoint the suggestion of returning RDF/XML subgraphs matching the query was not ideal, because the result had to be transformed in the invoking client to Jena Framework model for iterating the results feasibly. Also, from a plain text-search viewpoint it would be useful, if the services could be parameterized to return just the textual representations of the concepts. Such approach would be feasible for applications that are not ontology-aware, but could benefit from the ontological information. For example, classic text-based search engines could use services for gaining higher recall to the search engines queries.

[40]http://www.w3.org/2006/webapi/

6.3 Future Work

This study resolved the initial requirements and offered a proposal for ontology library system services from the end-user perspective. In the future the services should be evaluated by doing user tests from two viewpoints. First, the visualization of ontologies needs to be evaluated. Second, the interfaces should be evaluated from application developers viewpoint to see what services are useful and how they are used, and most importantly, are the services adequate.

As described in chapter 5.1, the implemented service framework Onki, offers ontology services for such ontologies, that are configured in the system configuration at the server startup. This is adequate when there are only a few ontologies on the server, but it does not support dynamic service configuration for the uploaded ontologies. To enable easier ontology publication to the server, ontology library system should support online publication of ontologies. This could be realized by offering an interactive web form, which either prompts the configuration from the end-user or alternatively the system should resolve the configuration automatically.

Ontologies evolve over time [Sto04] and new versions of ontologies are published when errors in the modeled domain are fixed and new information is added in form of new concepts and relations between them. Visualization of changes between versions and browsing from one version to another was not considered in this thesis, although such information would be useful to users developing and maintaining ontologies. Providing version information visually as well as programmatically accessible version information is one future research track.

Although ontology development tools such as Jena and Protege support ontology persistence to relational databases, the tools are not mature compared to object-relational-mapping (ORM) tools used in conventional relational-database applications. Efficient ORM-frameworks, for example Hibernate[41] and tools mapping business-tier services to web services are driving forces for developing classic relational database application. Such tools are also needed for ontologies to facilitate semantic web application development.

The proposed solution was designed from the viewpoint that the system contains ontologies, which are browsed and utilized separately. However, when more ontologies are published, mapping them together using vocabulary defined in the OWL specification is needed to gain full benefit from the published ontologies. The more

[41]http://www.hibernate.org

such mappings exist, the more complicated will the visualization of interlinked ontologies become, and the problem of providing feasible ontology visualization should be tackled.

Provided programmatically accessible web services return format was implemented as RDF/XML serialized subgraphs of the ontology. Such approach is not convenient for the invoking client to process when compared to the SKOS API [DB04] suggestion on presenting the concepts using XML Schema which enables easy transformation of the messages to native language classes, regardless of the programming language as long as object oriented methods are applied. However, the approach used in the SKOS API does not suite generic ontologies, because the assumption of a fixed set of concept's properties can not be done. Therefore it should be resolved if the web services need to be developed individually for all ontologies, thus fulfilling the needs of application developers.

The initial experiences of the Javascript API suggest that the work on providing ready-to-use user interface components should be continued. First, from a web application viewpoint, but also from a desktop application viewpoint. For example, ontology library services could be offered as a library containing Java Swing or C# graphical widgets providing ontology visualization and integration point to external applications.

Finally, from an architectonic viewpoint, the idea of providing high configurability to a system implementing ontology services is suitable for ontologies that are developed using few properties. It is also sufficient, when the services required by ontology end-users are simple, such as browsing an ontology and providing basic search functionalities. When the system was tested using different ontologies and end-users, the need for different visualization, support for different ontology standards and annotation services raised. These requirements were realized by extending the core implementation by adding new features which could be adjusted by modifying the system configuration. However, maintainability of a system supporting different and ontology-specific use-cases is challenging and is prone to produce low quality design and code. Therefore the idea of a configurable multipurpose ontology services tool is not feasible when designing general ontology services for end-users. A better approach would be a system that provides a set of core services and interfaces which a developer can extend to the specific needs of his ontology.

The presented ontology services prototype Onki was implemented to gain experiences of the services. The focus was in implementing the use cases rather than in design and architecture. The next generation ontology services could be considered as a lightweight ontology services framework based on the concept of 'inversion of control' (IOC) [JF88]. IOC is based on the idea of minimizing dependencies between components in an application. The dependencies are minimized by configuring each component as an individual service, and the application is assembled from these services by configurating the application rather than coding it. For example, frameworks such as Spring[42], HiveMind[43] and PicoContainer[44] provide mechanisms for creating applications that are based on this idea. Using such approach it is possible to develop a set of core ontology services and interfaces described in this thesis. Such an architecture could respond well to the arising need from different ontologies when compared to the implemented architecture. Basic implementation for the services could also be provided, leaving complex implementation of services for the developer community involved with domain ontologies.

Acknowledgements

This thesis is part of the National Finnish Ontology Project (FinnONTO) 2003-2007[45], funded mainly by The National Technology Agency (Tekes) and a consortium of 37 companies and public organizations.

I want to thank all my colleagues in the Semantic Computing Research Group[46] at the University of Helsinki and Helsinki University of Technology for ideas and suggestions considering the implemented prototype and the thesis as a whole. Especially fellow researchers Thomas Häggström Tomi Kauppinen, Tuukka Ruotsalo and Arttu Valo deserve to be mentioned. Although most of the ideas born in that room were "off target" and in most cases had nothing to do with my thesis, those moments ignited the spark for new ideas and offered occasional time off from the thesis writing process. Finally, and by no means least, I would like to express gratitude to my supervisor, Professor Eero Hyvönen, for the opportunity to work in the FinnONTO-project and most importantly for the guidance in getting the thesis done.

[42]http://www.springframework.org/
[43]http://hivemind.apache.org/
[44]http://www.picocontainer.org/
[45]http://www.seco.tkk.fi/projects/finnonto/
[46]http://www.seco.tkk.fi

References

ABG04 Aitchison, J., Bawden, D. Gilchrist, A., *Thesaurus Construction and Use: A Practical Manual, 3th edition.* Routledge, 2004.

BEK+00 Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S. Winer, D., *Simple Object Access Protocol (SOAP) 1.1.* W3C, 2000. URL `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`.

BG05 Brickley, D. Guha, R., *RDF Vocabulary Description Language 1.0: RDF Schema.* W3C, 2005. URL `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

BKvH02 Broekstra, J., Kampman, A. van Harmelen, F., Sesame: A generic architecture for storing and querying rdf and rdf schema. *Proceedings of the 1st International Semantic Web Conference (ISWC 2002).* ACM Press, 2002, 54–69.

BL05 Berners-Lee, T., *Uniform Resource Identifier (URI): Generic Syntax*, 2005. URL `http://www.gbiv.com/protocols/uri/rfc/rfc3986.html`.

BLHL01 Berners-Lee, T., Hendler, J. Lassila, O., The semantic web. *Scientific American*, 284,5(2001), 34–43.

BMR+96 Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. Stal, M., *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns.* John Wiley & Sons, 1996.

BT04 Binding, C. Tudhope, D., Kos at your service: Programmatic access to knowledge organisation systems. *Journal of Digital Information*, 4,4(2004).

BT05 Binding, C. Tudhope, D., Towards terminology services: experiences with a pilot web service thesaurus browser. *Proceedings of the International Conference on Dublin Core and Metadata Applications*, 2005, 269–273.

BvHH+04 Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. Stein, L. A., *OWL Web Ontology*

Language Reference. W3C, 2004. URL http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

CCMW01 Christensen, E., Curbera, F., Meredith, G. Weerawarana, S., *Web Services Description Language*. W3C, Version 1.1, 2001. URL http://www.w3.org/TR/2001/NOTE-wsdl-20010315/.

DB04 Dave Beckett, Nikki Rogers, A. M., *The Simple Knowledge Organisation System (SKOS) API: SKOSThesaurus*, 2004. URL http://www.w3.org/2001/sw/Europe/reports/thes/api/docs/org/w3/y2001/sw/Europe/skos/SKOSThesaurus.html.

DF01 Ding, Y. Fensel, D., Ontology library systems: The key to successful ontology reuse. *Proceedings of SWWS'01, The first Semantic Web Working Symposium*. Springer Verlag, 2001, 93–112.

DWM01 Das, A., Wu, W. McGuinness, D. L., Industrial strength ontology management. *Proceedings of SWWS'01, The first Semantic Web Working Symposium*. Springer-Verlag, 2001, 17–37.

EMSS00 Erdmann, M., Maedche, A., Schnurr, H. Staab, S., From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. *Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, 2000.

Fen01 Fensel, D., *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2001.

FFR97 Farquhar, A., Fikes, R. Rice, J., The ontolingua server: a tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, 46,6(1997), 707–727.

Fie00 Fielding, R. T., *Architectural Styles and the Design of Network-based Software Architectures, PhD Thesis.* , University of California, 2000.

FSvH03 Fluit, C., Sabou, M. van Harmelen, F. *Supporting User Tasks through Visualisation of Light-weight Ontologies*, Handbook on Ontologies in Information Systems. Springer-Verlag, 2003.

GB05 Grady Booch, Ivar jacobson, J. R., *Unified Modeling Language Reference Manual, 2nd Edition.* Addison Wesley, 2005.

Gru93	Gruber, T. R., A translation approach to portable ontology spesifications. *Knowledge Acquisition*, 5,2(1993), 199–220.
HM06	Hyvönen, E. Mäkelä, E., Semantic autocompletion. *Proceedings of the first Asia Semantic Web Conference (ASWC 2006), Beijing*. Springer-Verlag, 2006.
HS02	Handschuh, S. Staab, S., Authoring and annotation of web pages in cream. *WWW '02: Proceedings of the 11th international conference on World Wide Web*. ACM Press, 2002, 462–473.
HVK$^+$05	Hyvönen, E., Valo, A., Komulainen, V., Seppälä, K., Kauppinen, T., Ruotsalo, T., Salminen, M. Ylisalmi, A., Finnish national ontologies for the semantic web - towards a content and service infrastructure. *Proceedings of International Conference on Dublin Core and Metadata Applications (DC 2005)*, 2005.
IJ04	Ian Jacobs, N. W., *Web Services Architecture*. W3C, 2004. URL `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`.
JF88	Johnson, R. E. Foote, B., Designing reusable classes. *Journal of Object-Oriented Programming*, 1,2(1988), 22–35.
JKN01	Järvelin, K., Kekäläinen, J. Niemi, T., Expansiontool: Concept-based query expansion and construction. *Information Retrieval*, 4,3-4(2001), 231–255.
KH04	Korpilahti, T. Hyvönen, E., An architecture for collaborative ontology library development. *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI2004), Workshop on Application of Semantic Web Technologies to Web Communities*, 2004.
KH06	Kauppinen, T. Hyvönen, E. *Modeling and Reasoning about Changes in Ontology Time Series*, Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems, 319–338. Springer-Verlag, 2006.
KM02	Koivunen, M. Miller, E., W3C semantic web activity. *Proceedings of the Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications*. Helsinki Institute for Information Technology, HIIT Publications, 2002, 27–44.

KPO$^+$05 Kiryakov, A., Popov, B., Ognyanoff, D., Manov, D. Kirilov, A., Semantic annotation, indexing, and retrieval. *Journal of Web Semantics*, 10,2(2005), 39–78.

KVH05 Komulainen, V., Valo, A. Hyvönen, E., A tool for collaborative ontology development for the semantic web. *Proceedings of International Conference on Dublin Core and Metadata Applications (DC 2005)*, 2005.

LRP95 Lamping, J., Rao, R. Pirolli, P., A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems.* ACM Press, 1995, 401–408.

McG01 McGuinness, D. L. *Ontologies Come of Age*, The Semantic Web: Why, What, and How, 98–99. MIT Press, 2001.

MHSV04 Mäkelä, E., Hyvönen, E., Saarela, S. Viljanen, K., Ontoviews - a tool for creating semantic web portals. *Proceedings of the International Semantic Web Conference.* Springer-Verlag, 2004, 797–811.

MM04 Menola, F. Miller, E., *RDF Primer.* W3C, 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

NSD$^+$01 Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W. Musen, M. A., Creating semantic web contents with protege. *IEEE Intelligent Systems*, 16,2(2001), 60–71.

OVSM04 Oberle, D., Volz, R., Staab, S. Motik, B. *An Extensible Ontology Software Environment*, Handbook on Ontologies in Information Systems, 299–320. Springer-Verlag, 2004.

Pau05 Paulson, L. D., Building rich web applications with ajax. *IEEE Computer*, 38,10(2005), 14–17.

PE03 P. Eklund, R. Cole, N. R. *Retrieving and Exploring Ontology-based Information*, Handbook on Ontologies in Information Systems. Springer-Verlag, 2003.

PL03 Perrey, R. Lycett, M., Service-oriented architecture. *SAINT Workshops.* IEEE Computer Society, 2003, 116–119.

PS05	Prud'hommeaux, E. Seaborne, A., *SPARQL Query Language for RDF.* W3C, 2005. URL `http://www.w3.org/TR/2006/WD-rdf-sparql-query-20061004/`.
RO85	Rzepka, W. Ohno, Y., Requirements engineering environments: Software tools for modeling user needs - guest editors' introduction. *IEEE Computer*, 18,4(1985), 9–12.
Sev03	Sevcenko, M., Online presentation of an upper ontology. *Proceedings of Znalosti 2003*, 2003.
SGD04	Soualmia, L. F., Golbreich, C. Darmoni, S. J., Representing the mesh in owl: Towards a semi-automatic migration. *Proceedings of the KR 2004 Workshop on Formal Biomedical Knowledge Representation.* CEUR-WS.org, 2004, 81–87.
Som00	Sommerville, I., *Software Engineering.* Addison Wesley, 2000.
Sto04	Stojanovic, L., *Methods and Tools for Ontology Evolution, PhD Thesis.* , Universität Karlsruhe, 2004.
VCFLGP03	Vega, J. C. A., Corcho, Ó., Fernández-López, M. Gómez-Pérez, A., Webode in a nutshell. *AI Magazine*, 24,3(2003), 37–47.
VH06	Valkeapää, O. Hyvönen, E., A browser-based tool for collaborative distributed annotation for the semantic web. *5th International Semantic Web Conference, Semantic Authoring and Annotation Workshop*, 2006.
VOSM03	Volz, R., Oberle, D., Staab, S. Motik, B., Kaon server - a semantic web management system. *Alternate Track Proceedings of the Twelfth International World Wide Web Conference, WWW2003.* ACM Press, 2003.
WGA05	Wood, D., Gearon, P. Adams, T., Kowari: A platform for semantic web storage and analysis. *Proceedings of the 14th International WWW Conference.* Springer-Verlag, 2005.

Appendix 1. An Example Onki Configuration

As described in chapter 5.1, Onki provides an xml-configuration file to adjust the services framework for different ontologies. Following high-level parameters can be used to customize the provided services for various purposes. The example configuration describes detailed configuration alternatives of each element.

- `Repository`-element is used for specifying the data access to ontologies, which can be stored in either database, file or HTTP address.
- `Annotation-repository`-element provides a mechanism for storing annotations in the system.
- `Rule-engine`-element configures the use of external rule-storage for the ontology.
- `Classes`-element contains parameters for setting the visualizes concepts and their properties.
- `Result-order`-element is used for sorting the visualized concepts and properties.
- `Visualization`-element specifies what information of the concepts is shown to the user.
- `Utils`-element is an extension point for different utility tools for enhancing queries. For example a synonym-backend can be configured.
- `Login`-element enables authentication and user management to the system.

```
<?xml version="1.0" encoding="UTF-8" ?> <onki-services-conf>

    <!-- === 1. Set up connection to used ontologies ===================== -->
    <!-- impl:                                                             -->
    <!--   File(s) : fi.helsinki.cs.seco.onki.logic.impl.FileConnector     -->
    <!--   Jena DB : fi.helsinki.cs.seco.onki.logic.impl.JenaDbConnector   -->
    <!--   Http    : fi.helsinki.cs.seco.onki.logic.impl.HttpConnector     -->
    <!--                                                                   -->
    <!-- model:                                                            -->
    <!--   Look for static fields in com.hp.hpl.jena.ontology.OntModelSpec. -->
    <!--   Prefer non-inference models.                                    -->
```

```
<!-- ============================================================ -->
<repository impl="fi.helsinki.cs.seco.onki.logic.impl.FileConnector"
            model="OWL_FULL_MEM">

<!-- === 1.1 Jena Database Configuration ========================== -->
<!-- driver:                                                        -->
<!--   JDBC-driver classname (remember to put in lib/).             -->
<!-- url:                                                           -->
<!--   jdbc-url (jdbc:protocol://server/database).                  -->
<!-- username:                                                      -->
<!--   Database username/login.                                     -->
<!-- password:                                                      -->
<!--   Corresponding password for the username.                     -->
<!-- engine:                                                        -->
<!--   One of (MySQL, PostgreSQL, Oracle).                          -->
<!-- ============================================================== -->
<database>
    <driver>com.mysql.jdbc.Driver</driver>
    <url>jdbc:mysql://localhost/ONKI_VAO</url>
    <username>root</username>
    <password>root</password>
    <engine>MySQL</engine>
</database>

<!-- === 1.2 Ontologies in file(s) configuration ================== -->
<!--   Can have 1..n <include>/path/to/file.owl</include> elements. -->
<!-- ============================================================== -->
<file>
    <include>webapps/onki/samples/YSO-UO.owl</include>
</file>

<!-- === 1.3 Ontologies in HTTP-repository ======================== -->
<!--   Can have 1..n <url>/path/to/file.owl</url> elements.         -->
<!-- ============================================================== -->
<http>
    <url>http://localhost:8081/YSO.rdfs</url>
</http>
</repository>
```

```
<!-- === 2. Set up connection to an annotation repository ============ -->
<!--   For external clients to upload/insert annotations, which can be -->
<!--   browsed and viewed in this Onki-browser.                        -->
<!--   At the moment only persistent annotation repositories is used   -->
<!--                                                                   -->
<!-- enabled:    (boolean true/false)                                  -->
<!--   boolean (false, true) : is the annotation repository used       -->
<!-- ================================================================= -->
<annotation-repository enabled="false"
                       impl="fi.helsinki.cs.seco.onki.logic.impl.JenaDbConnector"
                       model="OWL_DL_MEM">
<database>
    <driver>com.mysql.jdbc.Driver</driver>
    <url>jdbc:mysql://wrk-2.seco.hut.fi/onki</url>
    <username>onki</username>
    <password>onki</password>
    <engine>MySQL</engine>
</database>
</annotation-repository>

<!-- === 3. Jena Rule-engine to derive more statements to model ====== -->
<!--   Makes it possible to use external rule-file for generating      -->
<!--   more statements to the underlying ontology model.               -->
<!--                                                                   -->
<!-- enabled:   (boolean true/false)                                   -->
<!--   is the rule-engine used or not.                                 -->
<!-- input:     (one element)                                          -->
<!--   The location of rules as URL.                                   -->
<!-- ================================================================= -->
<rule-engine enabled="false">
    <input>file:conf/onki.rules</input>
</rule-engine>

<!-- === 4. Define used root classes and properties as well as labels = -->
<!--   Configures the root classes and properties that are used in      -->
<!--   rendering hierarchies. Also defines labels that are shown        -->
<!--   from the concepts. Possibility to restrict the browser of        -->
<!--   showing only concepts that have certain metaclass.               -->
<!-- ================================================================== -->
```

```
<classes>

<!-- === 4.1 Property used in building hierarchy ================== -->
<!--   The property which creates the class-hierarchy.            -->
<!-- ============================================================ -->
<follow>http://www.w3.org/2000/01/rdf-schema#subClassOf</follow>

<!-- === 4.2 Root concepts of the ontology ======================== -->
<!--   Roots-node can have 1..nc root-elements.                   -->
<!-- ============================================================ -->
<roots mode="EXACT">
    <root>http://www.cs.helsinki.fi/group/seco/ns/2004/04/YSO#endurant</root>
    <root>http://www.cs.helsinki.fi/group/seco/ns/2004/04/YSO#perdurant</root>
    <root>http://www.cs.helsinki.fi/group/seco/ns/2004/04/YSO#abstrakti</root>
</roots>

<!-- === 4.3 Labels of concepts =================================== -->
<!--   Labels-node can have 1..n label-elements.                  -->
<!--                                                              -->
<!--   Having multiple labels is useful, if browser is constructed -->
<!--   from many ontologies, which have different label describing -->
<!--   the human readable label.                                  -->
<!-- ============================================================ -->
<labels>
    <label>http://www.cs.helsinki.fi/group/seco/ns/2004/04/YSO#label</label>
    <label>http://yso.fi/mao#label</label>
    <label>http://www.cs.helsinki.fi/group/seco/ns/2004/04/YSO#label</label>
    <label>http://www.cs.helsinki.fi/group/seco/ns/2004/05/VAO#label</label>
    <label>http://www.w3.org/2000/01/rdf-schema#label</label>
</labels>

<!-- === 4.4  Restriction on NOT showing certain concepts ========== -->
<!-- enabled:    (boolean true/false)                             -->
<!--   Is the restriction used or not.                            -->
<!-- predicate:                                                   -->
<!--   predicate defining the metaclass (rdf:type).               -->
<!--                                                              -->
<!--   allowed-types node can have 1..n type-elements.            -->
<!-- ============================================================ -->
```

```
<allowed-types enabled="false"
               predicate="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">
    <type>http://yso.fi/YSO#YSOconcept</type>
    <type>http://yso.fi/YSO#YSOgroupConcept</type>
</allowed-types>

<!-- === 4.4 Alphabetical index configuration ==================== -->
<!--   Alphabetical concept index is generated from this           -->
<!--   comma separated list.                                        -->
<!-- ============================================================== -->
<alpha-index>A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z</alpha-index>
</classes>

<!-- === 5. Define how the concepts and properties are sorted ======== -->
<!--   In most configurations, the properties describing the human     -->
<!--   readable label of the concept is in the predicate-element.      -->
<!--                                                                    -->
<!--   Both concepts and properties predicates-node can have           -->
<!--   1..n predicate-elements.                                         -->
<!--   It is possible to write custom sorting algorithm, as long as it -->
<!--   implements the Comparator-interface in Java.                     -->
<!-- ================================================================= -->
<result-order>
<concepts
    impl="fi.helsinki.cs.seco.onki.logic.impl.ConceptAlbhabethicalComparator">
    <predicates>
        <predicate>http://yso.fi/YSO#prefLabel</predicate>
        <predicate>http://www.w3.org/2000/01/rdf-schema#label</predicate>
    </predicates>
    <ascending>true</ascending>
</concepts>

<properties
    impl="fi.helsinki.cs.seco.onki.logic.impl.PropertiesAlbhabethicalComparator">
    <predicates>
        <predicate>http://www.w3.org/2000/01/rdf-schema#label</predicate>
        <predicate>http://yso.fi/#label</predicate>
    </predicates>
    <ascending>true</ascending>
```

```
</properties>
</result-order>

<!-- === 6. Define external plug-ins, which can be used in searching ==== -->
<!--   Ideas for external connectors, which can be used in searching.     -->
<!--   However, none of these have been implemented yet.                  -->
<!-- ================================================================== -->
<utils>
<synset enabled="false" impl="fi.cs.helsinki.seco.onki.utils.impl.SimpleSynset"/>
<speller enabled="false" impl="fi.cs.helsinki.seco.onki.utils.impl.SimpleSpeller"/>
<lexical enabled="false" impl="fi.cs.helsinki.seco.onki.utils.impl.SimpleLexor"/>
</utils>

<!-- === 7. Define visualization of ontologies ======================= -->
<!--   Defines visualization of the browser.                          -->
<!-- ================================================================ -->
<visualization>

<!-- === Define available locales and default locale for the UI === -->
<!--   ResourceBundles containing the locales are stored in         -->
<!--   WEB-INF/classes/OnkiLocales_**.properties. Each locale       -->
<!--   should have own bundle.                                      -->
<!--                                                                -->
<!-- id:                                                            -->
<!--    language-code for the locale, OnkiLocales_id-file should    -->
<!--    contain the localization.                                   -->
<!-- value:                                                         -->
<!--   Locale name shown in the UI.                                 -->
<!-- ============================================================ -->
<locales default="fi">
    <locale id="fi">Suomi</locale>
    <locale id="en">English</locale>
    <locale id="se">Svenska</locale>
</locales>

<!-- === External file containing labels for properties =========== -->
<!--   For example, one could want to use localized label for       -->
<!--   rdfs:label. These are read from conf/onki-visualization.rdf. -->
<!--                                                                -->
```

```
<!-- enabled     boolean (true/false)                                 -->
<!--   Are external label-"ontology used or not.                      -->
<!-- ============================================================== -->
<external-labels enabled="true"/>

<!-- === Stripping of long concept labels ========================= -->
<!-- enabled:     boolean (true/false)                                -->
<!--   Is stripping of long labels used or not.                       -->
<!-- Value:                                                           -->
<!--   Integer of the maximum length.                                 -->
<!-- ============================================================== -->
<strip-long-labels enabled="true">20</strip-long-labels>

<!-- === Show namespace for concepts and properties in the UI ===== -->
<!-- concepts   (boolean true/false)                                  -->
<!--   Is the namespace shown for the concepts.                       -->
<!-- properties (boolean true/false)                                  -->
<!--   Is the namespace shown for the properties.                     -->
<!-- ============================================================== -->
<show-namespace concepts="true" properties="true"/>

<!-- === Paging of large search results and alphabetical index ==== -->
<!-- enabled    (boolean true/false)                                  -->
<!--   Is paging enabled or not.                                      -->
<!-- value      (Integer)                                             -->
<!--   Is the namespace shown for the properties.                     -->
<!-- ============================================================== -->
<result-paging enabled="true">90</result-paging>

<!-- === Restriction of concept's properties view ================== -->
<!--   Restricting the shown properties of concepts based on the     -->
<!--   end user's knowledge of ontologies. The view could be changed -->
<!--   from the user interface.                                       -->
<!--                                                                  -->
<!--   roles:novice and expert. For novice only the selected are     -->
<!--   shown and for others all are shown.                            -->
<!-- =============================================================== -->
<novice>
    <property>rdfs:label</property>
```

```
        <property>rdfs:comment</property>
        <property>rdfs:subClassOf</property>
        <property>YSO:actuality</property>
    </novice>
    </visualization>

    <!-- === 8. Define user accounts ======================================= -->
    <!--   User authentication                                               -->
    <!-- ================================================================== -->
    <login enabled="true"
        impl="fi.helsinki.cs.seco.onki.logic.impl.OnkiUserSimpleBackend">
        <user user="john" password="doe"/>
        <user user="foo" password="bar"/>
    </login>
</onki-services-conf>
```

Appendix 2. Onki WSDL Interface

Onki provides a SOAP-interface for querying underlying ontologies. All SOAP-methods return a string containing Base64-encoded subgraph matching the query. A description of these methods is listed below. For a complete reference of the interface, the WSDL-description is also included.

- `getConcept(String uri)` Returns the concept matching the parameter given uri.
- `getConceptsMatchingLabel(String search, String lang)` Returns concepts matching the given label and language.
- `getConceptsMatchingRegex(String regexp, String lang)` Returns concepts whose labels match the given regular expression and language.
- `getOntologies()` Returns all the ontologies in the system.
- `getOntologyProperties(String ns)` Returns all the properties in the given namespace.
- `getOntologyLabels(String uri)` Returns properties which are configured as concepts's labels in the parameter given ontology.
- `getRelatedConcepts(String uri, Integer dist)` Returns all the concepts which are linked with some property to the parameter given one. The parameter 'dist' specifies how far in terms of edges is traversed from the concept. For example, value 1 would return only such concepts which are directly related to it.
- `getRootConcepts(String ns)` Returns root concepts of the given ontology. Concepts which have the same namespace are considered to belong to the same ontology.
- `getSiblingConcepts(String uri)` Returns all the concepts which have the same superclass as the parameter given one.
- `getSubConcepts(String uri, Boolean transitive)` Returns all subclasses or instances of the parameter given concept. Either direct or transitive concepts are returned depending on the boolean parameter.
- `getSubConceptsMatching(String search, String uri, Boolean begin, Boolean end, String lang)` Returns all the concepts which are subclasses or instances of the parameter given one. The label must match the parameter 'search' in the language 'lang'. The boolean parameters are used to add wildcards to the search string.
- `getSuperConcepts(String uri, Boolean transitive)` Returns all superclasses of the parameter given concept. Either direct or transitive classes are returned depending on the boolean parameter.

- `querySparql(String query)` Returns the subgraph matching the parameter given SPARQL-query.

```
<?xml version="1.0" encoding="UTF-8"?> <wsdl:definitions
    targetNamespace="http://localhost:8080/onki/services/OnkiService"
    xmlns:impl="http://localhost:8080/onki/services/OnkiService"
    xmlns:intf="http://localhost:8080/onki/services/OnkiService"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:message name="getSuperConceptsAllRequest">
    <wsdl:part name="uri" type="soapenc:string"/>
  </wsdl:message>

  <wsdl:message name="getSubClassesMatchingResponse">
    <wsdl:part name="getSubClassesMatchingReturn"
               type="soapenc:string"/>
  </wsdl:message>

  <wsdl:message name="getConceptResponse">
    <wsdl:part name="getConceptReturn" type="soapenc:string"/>
  </wsdl:message>

  <wsdl:message name="getSiblingConceptsResponse">
    <wsdl:part name="getSiblingConceptsReturn" type="soapenc:string"/>
  </wsdl:message>

  <wsdl:message name="getSiblingConceptsRequest">
    <wsdl:part name="uri" type="soapenc:string"/>
  </wsdl:message>

  <wsdl:message name="getConceptNodesRequest">
    <wsdl:part name="uri" type="soapenc:string"/>
  <wsdl:part name="length" type="xsd:int"/>
  </wsdl:message>
```

```
<wsdl:message name="getMatchingConceptsResponse">
  <wsdl:part name="getMatchingConceptsReturn"
             type="soapenc:string"/>
</wsdl:message>

<wsdl:message name="getMatchingConceptsRequest">
  <wsdl:part name="match" type="soapenc:string"/>
  <wsdl:part name="begin" type="xsd:boolean"/>
<wsdl:part name="end" type="xsd:boolean"/>
</wsdl:message>

<wsdl:message name="getSubConceptsRequest">
  <wsdl:part name="uri" type="soapenc:string"/>
  <wsdl:part name="direct" type="xsd:boolean"/>
</wsdl:message>

<wsdl:message name="getSuperConceptsResponse">
  <wsdl:part name="getSuperConceptsReturn" type="soapenc:string"/>
</wsdl:message>

<wsdl:message name="getSuperConceptsAllResponse">
  <wsdl:part name="getSuperConceptsAllReturn"
             type="soapenc:string"/>
</wsdl:message>

<wsdl:message name="getConceptRequest">
  <wsdl:part name="uri" type="soapenc:string"/>
</wsdl:message>

<wsdl:message name="getSuperConceptsRequest">
  <wsdl:part name="uri" type="soapenc:string"/>
</wsdl:message>

<wsdl:message name="getSubClassesMatchingRequest">
  <wsdl:part name="match" type="soapenc:string"/>
  <wsdl:part name="subClassOf" type="soapenc:string"/>
  <wsdl:part name="begin" type="xsd:boolean"/>
  <wsdl:part name="end" type="xsd:boolean"/>
  <wsdl:part name="xmlLang" type="soapenc:string"/>
```

```
</wsdl:message>

<wsdl:message name="getSubConceptsResponse">
  <wsdl:part name="getSubConceptsReturn" type="soapenc:string"/>
</wsdl:message>

<wsdl:message name="getConceptNodesResponse">
  <wsdl:part name="getConceptNodesReturn" type="soapenc:string"/>
</wsdl:message>

<wsdl:portType name="OnkiService">
  <wsdl:operation name="getConcept" parameterOrder="uri">
    <wsdl:input name="getConceptRequest"
                message="impl:getConceptRequest"/>
    <wsdl:output name="getConceptResponse"
                 message="impl:getConceptResponse"/>
  </wsdl:operation>

  <wsdl:operation name="getSubConcepts" parameterOrder="uri direct">
    <wsdl:input name="getSubConceptsRequest"
                message="impl:getSubConceptsRequest"/>
    <wsdl:output name="getSubConceptsResponse"
                 message="impl:getSubConceptsResponse"/>
  </wsdl:operation>

  <wsdl:operation name="getSiblingConcepts" parameterOrder="uri">
    <wsdl:input name="getSiblingConceptsRequest"
                message="impl:getSiblingConceptsRequest"/>
    <wsdl:output name="getSiblingConceptsResponse"
                 message="impl:getSiblingConceptsResponse"/>
  </wsdl:operation>

  <wsdl:operation name="getSuperConcepts" parameterOrder="uri">
    <wsdl:input name="getSuperConceptsRequest"
                message="impl:getSuperConceptsRequest"/>
    <wsdl:output name="getSuperConceptsResponse"
                 message="impl:getSuperConceptsResponse"/>
  </wsdl:operation>
```

```
    <wsdl:operation name="getSubClassesMatching"
                    parameterOrder="match subClassOf begin end xmlLang">
      <wsdl:input name="getSubClassesMatchingRequest"
                  message="impl:getSubClassesMatchingRequest"/>
      <wsdl:output name="getSubClassesMatchingResponse"
                   message="impl:getSubClassesMatchingResponse"/>
    </wsdl:operation>

    <wsdl:operation name="getMatchingConcepts" parameterOrder="match begin end">
      <wsdl:input name="getMatchingConceptsRequest"
                  message="impl:getMatchingConceptsRequest"/>
      <wsdl:output name="getMatchingConceptsResponse"
                   message="impl:getMatchingConceptsResponse"/>
    </wsdl:operation>

    <wsdl:operation name="getConceptNodes" parameterOrder="uri length">
      <wsdl:input name="getConceptNodesRequest"
                  message="impl:getConceptNodesRequest"/>
      <wsdl:output name="getConceptNodesResponse"
                  message="impl:getConceptNodesResponse"/>
    </wsdl:operation>

    <wsdl:operation name="getSuperConceptsAll" parameterOrder="uri">
      <wsdl:input name="getSuperConceptsAllRequest"
                  message="impl:getSuperConceptsAllRequest"/>
      <wsdl:output name="getSuperConceptsAllResponse"
                   message="impl:getSuperConceptsAllResponse"/>
    </wsdl:operation>
  </wsdl:portType>

<wsdl:binding name="OnkiServiceSoapBinding" type="impl:OnkiService">
  <wsdlsoap:binding style="rpc"
                    transport="http://schemas.xmlsoap.org/soap/http"/>

  <wsdl:operation name="getConcept">
     <wsdlsoap:operation soapAction=""/>
     <wsdl:input name="getConceptRequest">
        <wsdlsoap:body use="encoded"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://ws.onki.seco.cs.helsinki.fi"/>
   </wsdl:input>
   <wsdl:output name="getConceptResponse">
      <wsdlsoap:body use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://ws.onki.seco.cs.helsinki.fi"/>
   </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getSubConcepts">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="getSubConceptsRequest">
<wsdlsoap:body use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://ws.onki.seco.cs.helsinki.fi"/>
   </wsdl:input>
   <wsdl:output name="getSubConceptsResponse">
      <wsdlsoap:body use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://ws.onki.seco.cs.helsinki.fi"/>
   </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getSiblingConcepts">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="getSiblingConceptsRequest">
      <wsdlsoap:body use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://ws.onki.seco.cs.helsinki.fi"/>
   </wsdl:input>
   <wsdl:output name="getSiblingConceptsResponse">
      <wsdlsoap:body use="encoded"
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="http://ws.onki.seco.cs.helsinki.fi"/>
   </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getSuperConcepts">
   <wsdlsoap:operation soapAction=""/>
```

```
      <wsdl:input name="getSuperConceptsRequest">
         <wsdlsoap:body use="encoded"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://ws.onki.seco.cs.helsinki.fi"/>
      </wsdl:input>
      <wsdl:output name="getSuperConceptsResponse">
         <wsdlsoap:body use="encoded"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://ws.onki.seco.cs.helsinki.fi"/>
      </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="getSubClassesMatching">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getSubClassesMatchingRequest">
         <wsdlsoap:body use="encoded"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://ws.onki.seco.cs.helsinki.fi"/>
      </wsdl:input>
      <wsdl:output name="getSubClassesMatchingResponse">
         <wsdlsoap:body use="encoded"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://ws.onki.seco.cs.helsinki.fi"/>
      </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="getMatchingConcepts">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getMatchingConceptsRequest">
         <wsdlsoap:body use="encoded"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://ws.onki.seco.cs.helsinki.fi"/>
      </wsdl:input>
      <wsdl:output name="getMatchingConceptsResponse">
         <wsdlsoap:body use="encoded"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://ws.onki.seco.cs.helsinki.fi"/>
      </wsdl:output>
   </wsdl:operation>
```

```
  <wsdl:operation name="getConceptNodes">
     <wsdlsoap:operation soapAction=""/>
     <wsdl:input name="getConceptNodesRequest">
        <wsdlsoap:body use="encoded"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                       namespace="http://ws.onki.seco.cs.helsinki.fi"/>
     </wsdl:input>
     <wsdl:output name="getConceptNodesResponse">
        <wsdlsoap:body use="encoded"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                       namespace="http://ws.onki.seco.cs.helsinki.fi"/>
     </wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="getSuperConceptsAll">
    <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getSuperConceptsAllRequest">
        <wsdlsoap:body use="encoded"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                       namespace="http://ws.onki.seco.cs.helsinki.fi"/>
      </wsdl:input>
      <wsdl:output name="getSuperConceptsAllResponse">
        <wsdlsoap:body use="encoded"
                       encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                       namespace="http://ws.onki.seco.cs.helsinki.fi"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="OnkiServiceService">
    <wsdl:port name="OnkiService"
             binding="impl:OnkiServiceSoapBinding">
      <wsdlsoap:address location="http://localhost:8080/onki/services/OnkiService"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

Appendix 3. Onki Javascript API

Onki Javascript API provides a lightweight integration method for web applications. These methods can be used by including the javascript from the Onki-server to the calling remote application. For example, an annotation application or a portal can benefit from these functions.

- `openOnki` is used for opening the whole Onki-browser for annotation purposes.
- `sendOnkiXMLQuery` provides an AJAX-bridge to Onki's text-based concept search.

```
// ======================================================================
// Version : $Id: onki-client.js,v 1.0 2005/05/16 10:28:42 vpkomula Exp $
// Author  : vpkomula
// Date    : 28.04.2006
//
// Description:
//
//  Ontology Library System Onki Javascript API for using the browser
//  from external web-applications. Examples of using Javascript API,
//  check http://demo.seco.tkk.fi/onki/mao/annotation/ for details.
// ======================================================================



// ======================================================================
// Public API-methods (Use these from external applications)
// ======================================================================



/**=====================================================================
 * Integrate to Onki-Browser using window referencing
 *
 * @param onkiUrl                 = Base
 * @param destinationFieldUriId   = Form field ID where URI is fetched
 * @param destinationFieldLabelId = Form field ID where Label is fetched
 * @param delimiter               = Delimiter used in Label and URI field,
 *                                  if multiple values are fetched
 * @param rangeRestriction        = Set the Range-restriction to query
 * ======================================================================
 */
function openOnki(onkiUrl, destinationFieldUriId,
                  destinationFieldLabelId, delimiter, rangeRestriction)


/**=====================================================================
 * Integrate to Onki-Browser using xmlHttpRequest and autocompletion
 *
 * @param onkiUrl                 = Base Onki-URL
 * @param destinationFieldUriId   = Form field ID where URI is fetched
 * @param destinationFieldLabelId = Form field ID where Label is fetched
 * @param delimiter               = Delimiter used in Label and URI field,
 *                                  if multiple values are fetched
 * @param rangeRestriction        = Set the Range-restriction to query
 * @param returnId                = Where to return the focus (html-anchor)
 * @param autocompleteId          = What is the id of the autocomplete layer
 * =====================================================================
 */
function sendOnkiXMLQuery(onkiUrl, key, uriFieldName, delimiter,
                          rangeRestriction, returnId,
                          autocompleteId)
```