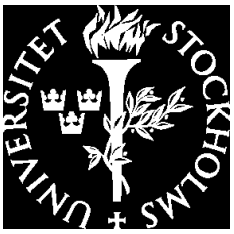

# Serverside Solution for Conceptual Browsing on the Semantic Web

**Fredrik Enoksson**

Master´s Thesis in Computer Science at Stockholm University, Sweden 2006


# Serverside Solution for Conceptual Browsing on the Semantic Web

**Fredrik Enoksson**

Master´s Thesis in Computer Science (20 credits)
Single Subject Courses
Stockholm University year 2006
Supervisor at Nada was Ambjörn Naeve
Examiner was Yngve Sundblad

TRITA-CSC-E 2006:040
ISRN-KTH/CSC/E–06/040–SE
ISSN-1653-5715

Department of Numerical Analysis and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

# Abstract

### Serverside Solution for Conceptual Browsing on the Semantic Web

This Master's thesis discusses how a server could provide information for a thin client Concept Browser, implemented for example on a mobile phone or as a Java Applet in a web browser. The core idea of a Concept Browser is to visualize structured information in the form of context-maps. When storing context-maps the language for the Semantic Web, RDF (Resource Description Framework), is utilized. For a thin client Concept Browser to be able to painlessly display a context-map, it is crucial to minimize the strain of loading and processing it. Hence, it is necessary to avoid the verbose RDF expression in XML as well as minimizing the information that is sent. The server side solution introduced in this thesis solves this by filtering out the information that is strictly necessary for a given context-map and then sends this information to the client over a specially designed protocol. This thesis will go through the requirements, design and implementation of the server as well as introduce a first version of the protocol.



# Sammanfattning

### Begreppsbrowser för den semantiska webben, en serverlösning

Denna rapport diskuterar hur en server kan tillhandahålla information för en begreppsbrowser som körs som en tunn klient, till exempel på en mobiltelefon eller en Java-applet i en webbrowser. Syftet med en begreppsbrowser är att visualisera information som kontext-kartor. När dessa kartor lagras så används beskrivnigs-språket för den semantiska webben, som kallas RDF (Resource Description Framework). För den begreppsbrowser som körs som en tunn klient ska kunna visa kontext-kartor utan problem så är det avgörande att undvika den datamängd som måste behandlas och överföras. Det är således nödvändigt att undvika den detaljerade och utförliga uttrycken i RDF, som serialiseras i XML samt minimera den information som sänds. Lösningen med en server som presenteras i denna rapport filtrerar ut den information som är absolut nödvändig för en speciell kontext-karta och skickar sedan denna information till klienten över ett protokoll designat för detta syfte. Rapporten går igenom kraven för servern och designen och implementationen av denna samt introducerar den första versionen av protokollet.

# Contents

# 1 Introduction

This work is a part of the distributive interactive learning environment being done at the Knowledge Management Research (KMR)[6] group, Centre of User oriented IT design (CID), Royal institute of technology (KTH), Stockholm.

Supervisors have been Ambjörn Naeve, Matthias Palmer and Mikael Nilsson and examiner Yngve Sundblad, CID, KTH.

## 1.1 The problem

At the KMR group an application called a Concept Browser [8] has been developed under the name Conzilla [9]. It is an application for displaying and navigating contexts. Each context is represented in the form of a context-map, which basically is a set of concepts and concept-relations. This application was mainly designed as an educational tool, but has proven useful in other situations as well. The latest version of Conzilla uses RDF [10], which is the language of the Semantic Web [13].

The initial purpose of this project was to make a version of Conzilla (or a Concept Browser) for the new mobile phones that have the possibilities to be programmed with the Java micro edition (J2ME)[4]. With a quick look at how Conzilla works with the information, it can be concluded that sometimes a lot of unnecessary information is loaded. Since these mobile phones have a limitation both in memory and bandwidth in connection with other devices it is necessary to reduce unnecessary information. The idea for a solution is to gather information to a proxy server first and then sort out the necessary information that is to be sent to the client.

The project was divided into two parts, one to investigate how the client could work and the other how the proxy server could work. This thesis deals with the latter and the goal with the project was to create a proxy server that could serve the thin client. To make it possible the following questions need to be answered:

- What could be requested of a proxy server to enable the idea of a Concept Browser as a thin client?
- What kind of protocol would be needed and how should it be designed?
- For these purposes, how should information from the Semantic Web be harvested and cached?

## 1.2 Method

To answer these questions it is necessary to understand what a Concept Browser is and how it represents contexts, concepts and concept-relations. From this it is possible to know what kind of information that is needed and how a protocol between the proxy server and the client should be designed. When this has been understood it is necessary to know how to harvest information and how to cache information for quick referencing. From this a design and an implementation could be made for the proxy server that could serve the thin client.

## 1.3 Report organization

*Chapter 2, Conceptual Browsing, a theoretical background* describes the principles of a Concept Browser and some necessary definitions closely connected to it.

*Chapter 3, Conceptual Browsing on the Semantic Web* describes how the Concept Browser could work with the Semantic Web and how the context-maps are described in RDF, which is the language for the Semantic Web.

*Chapter 4, Protocol for exchanging context-maps*, what kind of information is sent between the client and server and how it is composed is described in this chapter.

*Chapter 5, Proxy server for a Concept Browser*, describes the requirements for such a server and how the communication between the server and the client could be handled. This chapter also describes how a prototype for such a server was designed and implemented.

*Chapter 6, Conclusions* Summarizes the result and answers the problems for this project. Future perspectives for the proxy server are also discussed.

## 1.4 Definitions

The following definitions, taken from [8], are used in this paper:

- *Thing* = phenomenon or entity.
- *Concept* = representation of some thing.
- *Context* = graph containing concepts as nodes and concept-relations as arcs.

- *Context-map (or context diagram)* = graphic representation of a context.
- *Content (component)* = information linked to a concept or a concept-relation.

## 1.5 Abbreviations

The following abbreviations will be used in this thesis:
**CPU** Central Processing Unit
**CSS** Cascading Style Sheet
**EBNF** Extended Backus-Naur Form
**HTTP** Hypertext Transfer Protocol
**IP** Internet Protocol
**J2ME** Java Micro Edition
**MIME** Multipurpose Internet Mail Extensions
**PDA** Personal Digital Assistant
**RDF** Resource Description Framework
**RDFS** Resource Description Framework Schema
**TCP** Transmission Control Protocol
**URI** Uniform Resource Identifier
**URL** Uniform Resource Locator
**UML** Unified Modelling Language
**ULM** Unified Language Modelling
**XML** eXtendable Markup Language

# 2 Conceptual Browsing, a theoretical background

The increased use of information and communication technology has led to that the amount of information retrievable today is growing very rapidly. Even though the new technology has made it easier to find and collect information, it is sometimes hard to find the relevant information, especially when using the WWW. To understand what context the information is in can be difficult unless the website is carefully designed. Otherwise the feeling of 'Within what context am I viewing this content and how did I get here?' appears, also referred to as the websurfing sickness. Even if the system with hyperlinks on the WWW can prevent the websurfing sickness, the contextual topology for this system leads to some problems, which basically comes from the fact that the context and the content are not separated. A Concept Browser is a way of doing this, though a total separation of context and content is not made. In such a browser every context is displayed as a context-map, where a concept in that map can refer to content describing that particular concept. But before Conceptual Browsing is described any further a definition of contextual topology is needed.

## 2.1 Contextual topology

A contextual topology describes how concepts and their contexts are treated. A. Naeve's definition, taken from [7], is:

Let **S** be a set of concepts, and let $C$ be a concept in **S**. A context in **S** that contains $C$ is called the *contextual neighborhood* of $C$ in **S**. The *contextual topology* on **S** is the set of all contextual neighborhoods (in **S**) of concepts of **S**. If a concept $C$ has no contextual neighborhood involving other concepts from **S**, then $C$ is called an *isolated concept* in **S**.

This definition is influenced by the definition of a mathematical topology and is therefore theoretical. The following examples describe some contextual topologies currently used and their shortcomings.

### 2.1.1 Different kinds of Contextual Topologies

The contextual topologies used at a certain time in history depends on the technology of that time. Therefore different kinds of topologies has evolved from how it was possible and useful to form a structure for the content to be presented in. One basic way to form such a structure is to create a lexicographical order, like a dictionary. This is a simple way to find information, but with the disadvantage of not showing any relations between the content entries of the dictionary. Every entry in the dictionary is an isolated concept and forms a totally disconnected contextual topology. Encyclopaedias can

be viewed as dictionaries, but with the ability to refer to another entry (usually) in the same encyclopaedia much like a hyperlink system as the WWW, though in the closed world of an encyclopaedia. A textbook on the other hand usually makes some kind of classification or taxonomy of the information to create a context. A book about vehicles would classify a vehicle into the subtype motor vehicles and manforce vehicles. Further subtyping could create the context for the text (content) of the book. The context is built upon a taxonomic system.

With the Internet another contextual topology has evolved in the form of the WWW, which is basically a huge set of webpages that can be hyperlinked to each other. It is in a way a dynamic system, since these hyperlinks could quite easily be created or removed. A typical webpage is a container for its content, but it is also a context for the content that can be reached from the page using hyperlinking. If a typical webpage $A$ has a link to webpage $B$, then $A$ forms a context for $B$. The whole context for $B$ are the webpages that link to it. $B$ can also be linked to $A$ and then $B$ has formed a context for $A$ and the relation is reversed. This leads to a complex structure of context and content, where the context can be hard to grasp. The context that is formed with the link system usually depends more on catching the attention of the viewer than on showing the conceptual relation of the content.

The topologies described form a relation with the content and context in an unsatisfying way. In the system with hyperlinks, the context and content are presented at the same time, which makes it hard for the user to get an overview of the context. Considering the two other topologies for books, the dictionary forms a lot of isolated concepts. Even though the normal textbook that has a taxonomical structure for the context gives you a good overview, the relationship between the context and the content is fixed. And so is the overall context for the book. The reuse of content in another context is not possible after a book has been printed.

## 2.2 Principles for a Concept Browser

To avoid the discussed shortcomings a contextual topology that is both dynamic and gives a good overview of a certain context is needed. For that purpose the Concept Browser was created, where it is possible to navigate through different kinds of contexts and concepts. The ideas for a Concept Browser were described in [7] and the following principles were put up in that paper:

1. Separate the content of a concept or a concept-relation from its contexts. This supports the reuse of conceptual content across different contexts.

2. Describe each separate context in terms of a context-map, preferably expressed in the Unified Language Modelling technique.

3. Allow neighborhood-based contextual navigation on each concept and concept-relation by enabling the direct switch from its presently displayed context into any one of its contextual neighborhoods.

4. Assign an appropriate set of resources as the content components of each appropriate concept and/or concept-relation.

5. Label each resource (concept, concept-relation, context or content component) by making use of a standardized data description (= metadata) scheme.

6. Allow metadata based filtering of the content components through context-dependent aspect-filters. This enables the presentation of content in a way that depends on the context.

7. Allow the transformation of a content component, which is also a context-map, into a context (called contextualization).

8. Support lateral thinking by introducing a concept bookmaker, which allows concepts as well as contexts to be interactively constructed from content according to a menu of different content-gathering principles.

In this thesis a Concept Browser is an application that follows these principles. In the rest of this text when referring to the principles these are the principles of a Concept Browser unless stated otherwise. The principles are discussed more in section 4.1.

### 2.2.1 Contextual topology in a Concept Browser

Each context is represented as a context-map in a Concept Browser, according to principle 2. These maps includes concepts and concept-relations (which basically is a concept). A certain concept can occur in several maps and according to the definition those maps form the contextual neighbourhood for that concept. This means that the contextual topology in a Concept Browser is a set of context-maps. A comparison could be made with an atlas. Each map where Stockholm appears in the atlas can be seen as the contextual neighbourhood for Stockholm.

### 2.2.2 Context-maps

It is the context-maps that make Conceptual Browsing possible, because the Concept Browser enables the user to get an overview of a concept or a context. They are the most essential part of a Concept Browser and they enable a separation between context and content since the concepts

are assigned sets of descriptional contents (webpage, picture, etc.). In this way the context-map (and the Concept Browser) builds a layer on top of the already existing WWW. By forming this context layer a structure is created for it. A context-map is the graphical representation of a context.

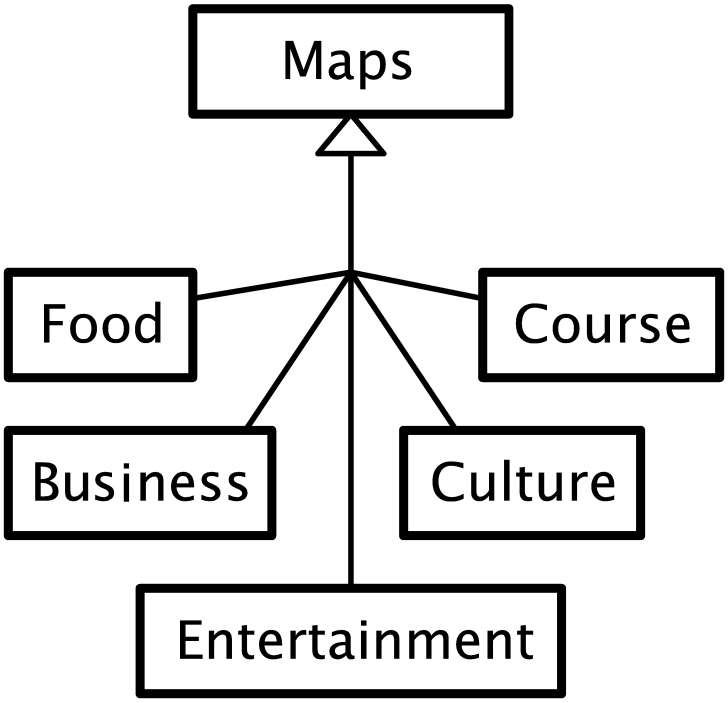


Figure 1: An example of a context-map.

The context is in turn a set of concepts related to each other by concept-relations. Such a relation is basically another concept which includes the concepts from the relation and the type of the relation. The context-map can be seen as a canvas where concepts and concept-relations are drawn. A *concept* is represented as a two-dimensional geometric entity (a square, circle, etc). The *concept-relations* are represented as lines between two or more concepts. Both the concepts and the concept-relations can be styled in different ways, preferably defined in advance. A good example of this is the class diagrams in UML, where concepts and concepts-relation are styled in different ways depending on their type. An example of a context-map is shown in figure 1.

## 2.3 The Concept Browser Conzilla

Conzilla is the first implementation of a Concept Browser. It was developed by Mikael Nilsson and Matthias Palmer, under the supervision of Ambjörn Naeve at CID/NADA/KTH. This application is based on the principles of a Concept Browser. It also has another feature for navigation through the context-maps. The basic navigation method through the concepts is done by doing a search of the contextual neighbourhood as described in principle 3. The other way of navigating implemented in Conzilla is by placing a hyperlink from a concept or concept-relation to a context-map, which is described in [9]. This feature could be used to link to any context-map, though it should be used to link to a map that describes the chosen concept in more detail. This feature will be considered further in this thesis, mainly because it is a good way of navigating through the different concepts.

Another useful feature of Conzilla is that it is possible to create and edit maps through a graphical interface. You are able to manipulate the style and hyperlinks of the concepts and concept-relations. Though this is a very powerful feature it will not be considered any further, since this thesis only deals with presenting maps.

# 3 Conceptual Browsing on the Semantic Web

The context-maps used by a Concept Browser should preferably be represented in a standardized way according to principle 5. In this thesis the choice of representation is Resource Description Framework (RDF), since the maps that existed were represented in that form. RDF is the language designed for the Semantic Web, which is an initiative from the World Wide Web Consortium (W3C). The visions of the Semantic Web is described in [13]. It is a vision of creating interoperability between information systems by forming a common semantic (understood by machines). There are several reasons for using RDF, the main one being that the Semantic Web and the Concept Browser could enhance each other. The following points mark out the advantages of using them together.

- ***W3C recommendation*** RDF is a the new metadata standard for interoperability between machines recommended by the W3C. Since it is an open standard the representation could be interpreted by other applications understanding RDF.
- ***Conceptual thinking*** The Semantic Web focuses on describing resources (a subset of those can be thought of as concepts) and relations between them. That is very similar to what is expressed in a context-map.
- ***Extensibility*** RDF is extendable, and therefore old metadata-descriptions could quite easily be translated into RDF. This way, context-maps represented in another language could be transformed into RDF and then used by an application like Conzilla.

## 3.1 RDF, a brief description

Resource Description Framework (RDF) is a language for describing information about Web resources. It is designed to represent information about things that can be identified on the Web, even if it is not retrievable there. For a resource to be identified on the WWW it has to have a Uniform Resource Identifier (URI).

The main data-model for RDF consists of three object types:

- ***Resources*** A resource is something that can be described with RDF. It is identified by an URI, therefore every resource is unique. Since (theoretically) everything can have a URI, it can be described by RDF.
- ***Properties*** A property is basically a resource, but it is a kind of resource that is used to describe characteristics, attributes or relations to other resources.

- ***Statements*** A resource can have a certain kind of property, and a value for that property. This is called a statement in RDF and it consists of three parts, the resource, which is called the subject of the statement, the property for this resource, called the predicate and the property value, which is called the object. So, a statement consists of a triple with a subject, a predicate and an object. The object of a statement can be another resource, but it can also be a literal, in other words a String. A statement is sometimes called an RDF-triple.

Every statement in RDF can be visualized as a graph, a resource is an ellipse (or a circle), the predicate an arrow pointing from the subject to the object. If the object is a literal it is depicted as a box. What has just been described is shown in figure 2 and 3.

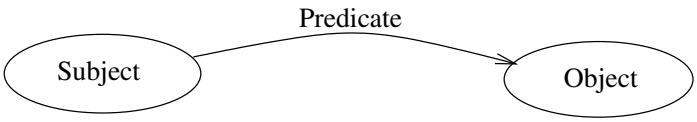


Figure 2: How a statement is visualized.

Consider a simple example, a webpage that has the URI *http://www.example.org/* has been created by Bruce Banner. In this case the webpage is the subject and it has got a property creator with the value Bruce Banner. The graph in figure 3 shows us the same thing.

To make the following examples a little easier to read, the URI will be abbreviated according to the following: *ex:* is short for *http://www.example.org/*, *rdf:* short for *http://www.w3.org/1999/02/22-rdf-syntax-ns#* and *rdfs:* for *http://www.w3.org/2000/01/rdf-schema#*. This would mean that *http://www.example.org/BruceBanner* is abbreviated *ex:BruceBanner*.

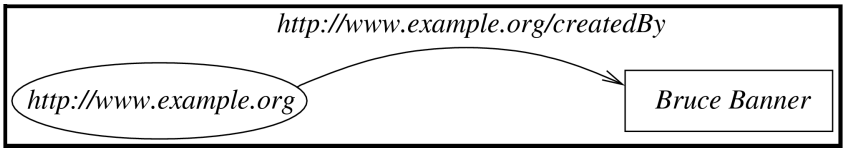


Figure 3: RDF-graph stating that http://www.example.org/ was created by Bruce Banner.

The object in the current example is a literal. Now, to include information about the creator's phone number and email address, we have to extend our description by replacing the literal with a resource. We can now let this

resource have the properties email, phone number and name. This is shown in figure 4. As seen in this last example, a resource can be the object in one statement and the subject in another. The advantage of doing as in the last example, is that you have a unique identifier for the object, which might not be the case in the first example, where we had a literal instead. If *ex:BruceBanner* is the creator of another webpage stated in another statement, we know that it is the same person.

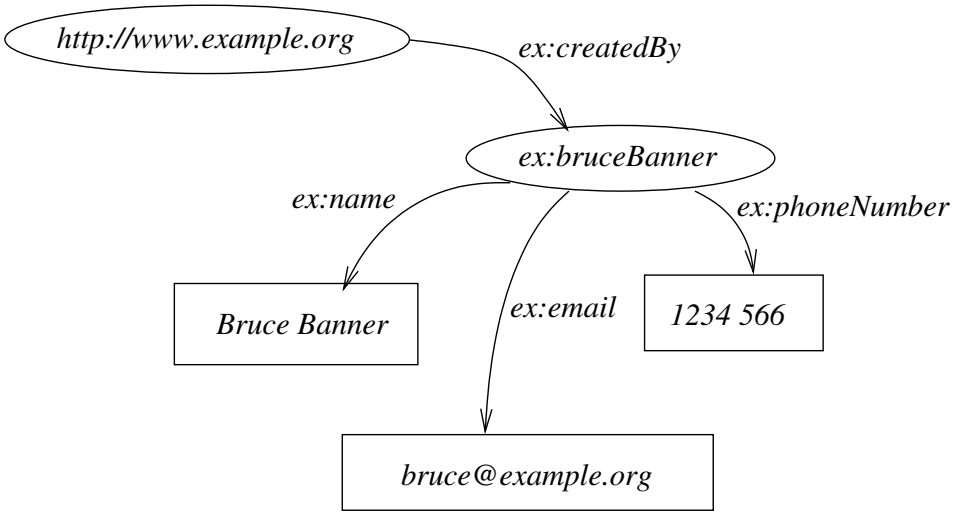


Figure 4: Extending the RDF-graph in figure 3.

### 3.1.1 RDF Classes and Vocabularies

Every resource in RDF can belong to, or be an instance, of one class or more. To state that a resource is of a certain class it is given the property *rdf:type*, with the class as value. The basic class in RDF is *rdfs:Class* defined in RDF Schema. With the help of this schema it is possible to form class hierarchies which have a lot in common with classes in object oriented programming languages like Java and C++. As a help to construct classes and their hierarchies RDF Schema provides a vocabulary using RDF for that purpose. A vocabulary is a set of properties and classes defined for a special purpose or organisation. RDF Schema is the base vocabulary and it is an extension to RDF that describes how to define your own vocabulary, by using RDF itself. This schema is like a type system for RDF which also provides mechanisms for how to specify your own classes and properties. As an example of RDF schema a class can be the subclass of another class with the help of the property *rdf:subClassOf*. Properties can also form a class hierarchy, that

is made with the property *rdf:subPropertyOf*. That and more about RDF classes is described in [11] and [10].

### 3.1.2 Reification

It is sometimes necessary to make statements about statements, for example who made a certain statement and when it was made. This can be done in RDF by using reification with the help of the class *rdf:Statement* and the three properties *rdf:subject*, *rdf:predicate* and *rdf:object*. An example of a reification is shown in figure 5 and it corresponds to one of the statements in figure 4. Through reification it is possible to have a statement both as the subject or the object of another statement.

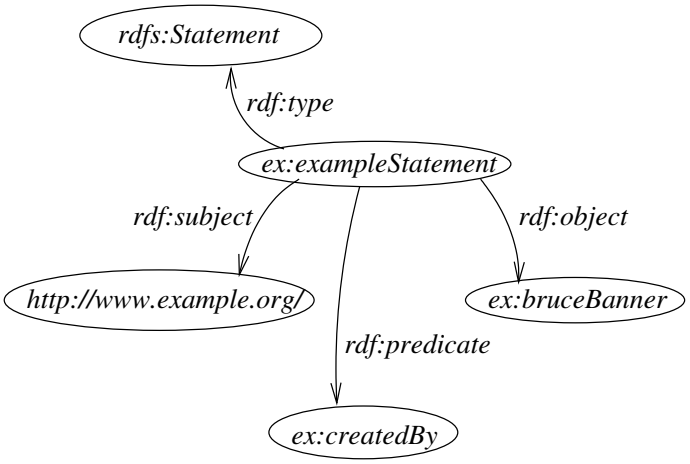


Figure 5: A reification of the statement in figure 4.

### 3.1.3 RDF serialized as XML

RDF can be serialized in many ways, but the most common way is to use XML. The XML-serialization for the example in figure 4 would be:

```
<rdf:Description rdf:about='http://www.example.org/'>
   <ex:createdBy rdf:resource='http://www.example.org/BruceBanner'>
</rdf:Description>

<rdf:Description rdf:about='http://www.example.org/BruceBanner'>
   <ex:name>Bruce Banner<ex:name>
   <ex:email>bruce@example.org<ex:email>
   <ex:phoneNumber>1234566<ex:phoneNumber>
</rdf:Description>
```

How RDF is serialized in XML is further described in [12]. The location where the RDF serialization is stored is called a container and when using XML a container is usually a file or a database. This means that several statements in RDF can be kept in the same container and information about one URI can be kept in different places as well. This causes some problem which will be discussed more in section 3.2.2 and 5.2.

## 3.2 Context-maps on the Semantic Web

To display a context in the form of a context-map it needs a graphical representation. Two cases could be considered, the context-map holds no information in advance and is styled by the application each time it is loaded or it could be styled in advance and the graphical information is stored with the map. Only the second case will be considered in this thesis, since if a map looks the same in every application it is more easily recognized as the same map by a user. Algorithms for layout usually works quite poorly and still use a lot of processing time. It is sometimes useful to enhance certain structure in a context-map, for example in a UML class diagram it is very common to place a superclass in a class hierarchy higher then its subclasses. The disadvantage is that the graphical information makes the maps use more memory, but taken the other advantages of it makes it better to store the graphical information. Another reason for this is discussed later on in this chapter.

For a Concept Browser the necessary information about the context-map and its concepts can be divided into four types:

- ***Abstract information*** The metadata about a concept, concept-relation or context-map.
- ***Graphical information*** If the map is not empty some concepts and relation between them are drawn on the map. This information describes how the map is drawn and can be used by an arbitrary application to present the map.
- ***Navigational information*** An example is the hyperlink to another map from a concept or concept-relation, as described in section 2.3. Indirectly the contextual neighbourhood is included here, as described in principle 3.
- ***Information about content*** Every concept or concept-relation can link to a resource that holds descriptional content about that certain concept. According to the design principle 4.

The context-maps and all information about them dealt with in this thesis are constructed in advance and are represented in RDF. Every map, concept

and concept-relation has an URI as its identifier. As described in section 3.1.3 this information is stored in containers, which can be a file or database of some kind holding RDF-statements. One container usually contains information about several URIs, therefore several context-maps can be included in one container. A context-map also needs information held by other URIs, which in turn can be held by another container. This means both that a container can hold several context-maps and that all information about a specific map can be distributed over many containers. Another thing is that a context-map rarely shows all the information in a container. One reason is that a container can hold a vast amount of information which would create a big context-map and the advantages of displaying it as a context-map would be lost. Another reason (closely related) would be that information would be too detailed and those details might not be relevant in certain situations.

### 3.2.1 The structure of context-maps

Since not all information in a container should be displayed two layers have been created. The bottom layer is called the information layer and on top of that is the presentational layer. The presentational layer is basically the graphical information of the context-map, but it refers to the content information and the information about a detailed map. With that is yet another layer called the style layer, where concepts and concept-relations could be styled in different styles according to some predefined scheme. For example a certain concept could be styled as a box with rounded corners another one as a circle, a concept-relation might have a dotted line or a special arrow at the end.

This structure could be compared with the structure of information in a server with dynamic webpages. The information is usually stored in an XML-file or a database. This information is glued together with the styling information to form an html-page and what is called dynamic webpages are created. On top of that some kind of stylesheet is sometimes included. These three layers are basically the same with context-maps. The context-map is referring to information in different containers and on top of that some kind of styling sheet is provided. All this according to figure 6. Two differences can be detected here. First, a dynamic webpage is constructed with the information and styling information mixed together. Second, the information is (usually) not available without the styling information. When using RDF everyone can see the information without the layout.

To represent a RDF-statement on the presentational layer every concept is represented by a concept layout and every relation by a statement layout. A concept layout holds the graphical information and also points to

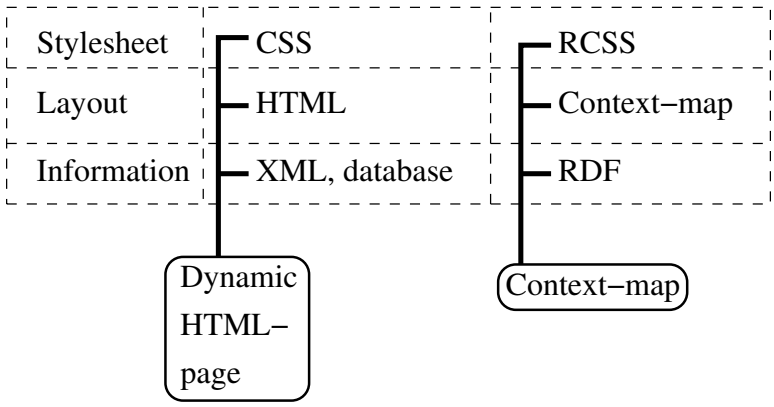


Figure 6: Comparing the gathering of information for dynamic HTML-pages and context-maps, however RCSS (RDF-CSS) is still a bit unspecified.

the content information and the link to the detailed map about that concept and most importantly it points to the concept in the information layer. The statement layout is built almost the same way, but the statement layout does not refer directly to the information layer, but to a reification of the statement which refers to the subject, predicate and object of the statement on the information layer. This is shown in figure 7. The statement layout also refers to the concept layout of the concept of the statement. The reason for this is that otherwise it would not be possible for the same concept to appear more than once in one map. The statement layout needs to know the layout of the concept it should be using or the layout could get wrong since several concept layouts can reference the same concept. For a statement layout to be correctly constructed the layout of the subject of the statement (a concept layout) should refer to the same as the subject of the reification of that statement layout. The same goes for the object and object layout, like in figure 7.

### 3.2.2 Conzilla Vocabulary

Within the Conzilla project a certain RDF-vocabulary for Conzilla has been developed to be able to represent context-maps. The URI http://kmr.nada.kth.se/rdf/graphic# is abbreviated *CVL:* in the following text. For a resource to be a context-map, the *rdf:type* has to be *CVL:ConceptMap*. All non-empty maps contain concepts and concept-relations, which have the type *CVL:ConceptLayout* and *CVL:StatementLayout* respectively. These correspond to the concept layout and the statement layout discussed in the previous section. With the help of the property *CVL:displayResource* both the layouts can indicate the URI of the resource it is styling. Parts of the Conzilla vocabulary is included in Appendix A.

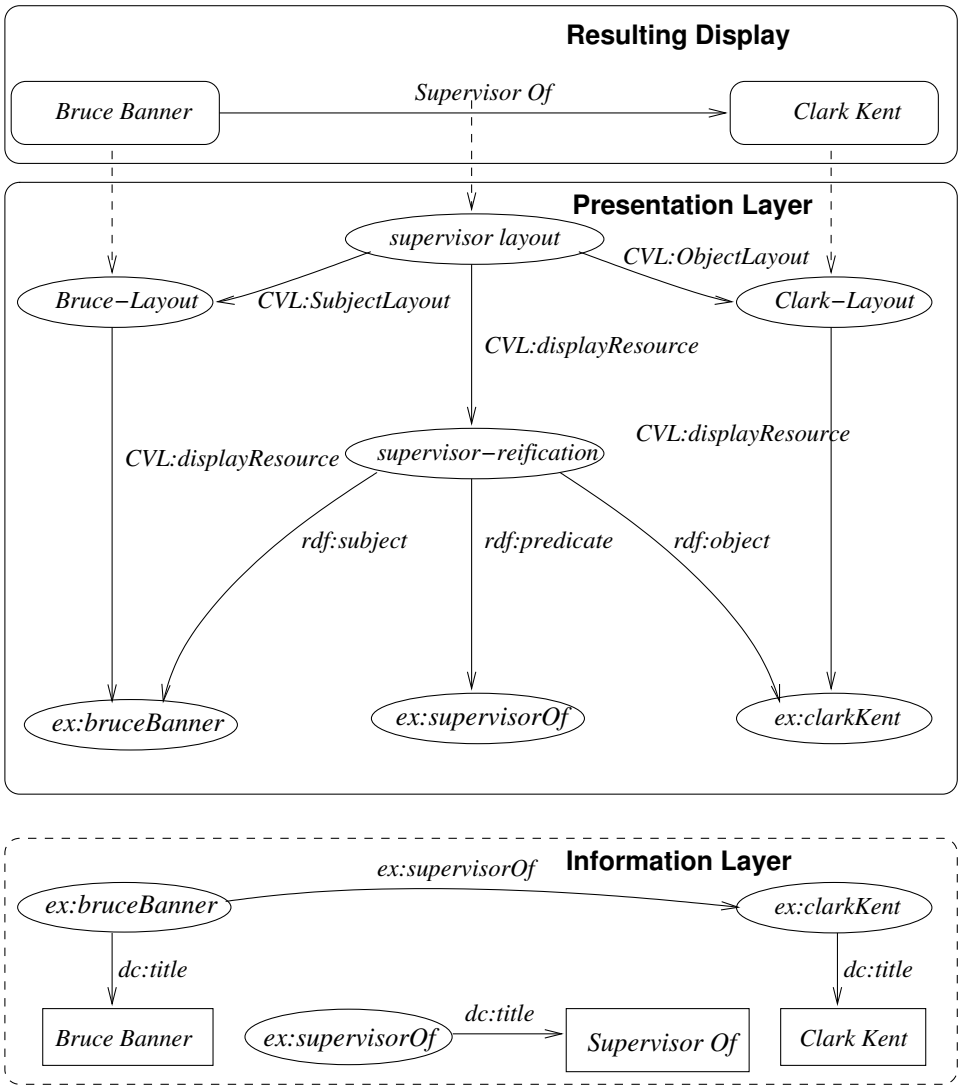


Figure 7: The layout parts refer to the information layer, the top part shows one way to display it.

One way of navigating through the contexts-maps is to do a contextual neighbourhood surf, as described earlier. Since only parts of the containers are displayed in a context-map the contextual neighborhood for a concept are the maps where a certain concept is displayed. A concept is identified by its URI and it is possible to find the contextual neighborhood of the concept only if it is displayed in a context-map. Also included in the Conzilla Vocab-

ulary are properties and Classes for navigation and location of context-maps. They all start with the URI *http://kmr.nada.kth.se/rdf/navigation#*, which will be abbrieviated *CVN:*. A styled resource is able to hyperlink to another context-map. To do this the property *CVN:hyperlink* is used for a concept layout or statement layout.

### 3.2.3 Locating and loading context-maps

Every map has a URI, which refers to other URIs, but the URI does not necessarily define where the container for that information is located. All information for a context-map might not even be included in one container. To be able to find in which container a context-map (or any other resource) is contained a property named *CVN:includeContainer* is sometimes included to indicate where the information is located. Unless any other method is provided for finding containers this is a good way to find it. Techniques how to solve this problem are discussed in section 5.2.

# 4 Protocol for exchanging context-maps

The thin client and the proxy server need a predefined way of communicating the necessary information between them. For a thin client to work as a Concept Browser there are some types of information that are more essential than other. This could be deduced from the principles of a Concept Browser.

## 4.1 Requirements for a Concept Browser Lightweight protocol

The first principle says that the content should be separated from its context and principle 2 to express these contexts in the form of a context-map. This is assumed to already be fulfilled, since all information is created in advance. The same goes for principle 4, which says to assign content components to a concept. Principle 5, that says that the concepts should be labelled with metadata is fulfilled as well and that choice is RDF as said earlier. Principle 6 is too advanced for a thin client and is not considered further. Principle 7 says that a content component could be another context-map. Principle 8, to enable a concept bookmarker, could be solved by the server. This would make the server save a lot of information about each and every client ever accessed it. Hence, it is better to keep those bookmarks on the client side. Taken this into consideration the principles of a Concept Browser boils down to 5 types of requests. The client would have the possibilities to request:

- A context-map. Every context is already in the form of a context-map, but to be able to display it the client should be able to get the information about them from the server.
- A contextual neighborhood. According to principle 3 it should be able to look at the contextual neighbourhood of a concept.
- Metadata, principle 5 says that each concept should be labelled with a standardized metadata description, which is RDF. Therefore it should be possible to request metadata about different resources. For example:
  - concept-metadata
  - metadata about a content component
- List of content components assigned to a concept. According to principle 4 content components could be assigned to a concept. Information about these should be able to request.

- A content component, if the client have the restriction of an unsigned Java Applet, to only be able to download things from the server the applet originated from. In this case it would be necessary to download the content to the proxy server first and then to the client. So requesting content should (to some extent) be possible. A content component could be a context-map and then it should be handled as downloading a context-map.

A protocol was designed for these 5 types of requests. The current working name for the protocol is Lightweight Concept Browser Protocol, abbrieviated LCP. It is a text protocol and the goals with the design were to make it:

- ***Simple and stateless*** Since this a first version it is good to keep it simple. Stateless because it is easier to implement such a protocol, cause one request demands only one response
- ***Compact*** From the structure of information discussed earlier and the limitations of the thin client it is important to reduce unnecessary information and overhead.
- ***Extendable when possible*** Though the protocol is designed to be extendable in this first version the two first points are more important in the design. But the design should make extensibility when possible if it does not interfere too much with the other two design goals.
- ***Not dependent on any vocabulary*** Since the maps are expressed in RDF different vocabularies for expressing context-maps can exist. The protocol for that should not be dependent of a certain vocabulary.

### 4.1.1 Is XML suitable?

The most common way to serialize RDF is to use the eXtandable Markup Language (XML). It would be possible to just cut out the important parts of the XML-serialization or create another XML-serialization and transmit that. This is of course possible but since XML is very expressive it would use a lot of bandwidth and probably a lot of memory on the client side. This could partly be solved by compressing it in some way, but that would instead require more CPU utilization.

### 4.1.2 Is binary suitable?

A compact format like a binary representation of the map could be used. That way a lot of bandwidth could be saved, though it would not be readable by a human and it would probably be hard to easily extend. A binary version could possibly use a lot of CPU-time if not carefully designed. Another

approach than binary is to rely on smart algorithms in the lower parts of the network layer to save bandwidth.

### 4.1.3 An intermediate way

By not using XML parts of the extensibility and flexibility is lost and by not using a binary format the protocol could end up using a lot of bandwidth. The proposed protocol uses an intermediate approach. In fact, the protocol is text based and balances between expressisivity and minimizing overhead information. That way a compromise between using too much memory, CPU-time and bandwidth is hopefully achieved.

## 4.2 Design of requests and responses

When a client requests the proxy server for information it needs to know what type of information that is requested. This means that each request should have an identifier for each type of request. Except this identifier at least one argument is included depending on the request, which will be discussed in this chapter. The structure for the request is constructed so that all necessary information for the server to consider is on the first line of the incoming text and it ends with LCP/ followed by the version number. After that additional information about the client could be included on new lines, for example screensize. This is done the same way as in HTTP [3] and it follows after the first line in the request. This could be included in every request, though it is up to the server to actually consider them any further. The request ends with two new lines which states the end of the request. This and the rest of the design of the request part of the protocol are influenced by the HTTP-protocol. The requests and response could be mixed up, for example if a request is sent before the previous request got its response. There is no way of knowing what response corresponds to which request. This will be handled by the underlying network protocol like TCP/IP, where a connection is defined by an address and a port number on each side of the connection. The design of LCP needs such a underlying to know what request responds to what response. Since the information of the client is rather sparse and no login is required the issue of session has been postponed. Though, by the use of HTTP as an underlying protocol that would be quite easily achieved through cookies or similar techniques.

Every request from a client should be answered with a corresponding response, except if something goes wrong. In that case an error message is sent instead. To recognize the protocol the response starts with the codeword LCP/ followed by the version number and the codeword for the request, which is followed with two new lines. For the client to know when all information is sent every response and request ends with two new lines. Note

that some of the examples in the following chapters could not be fit into one line. A backslash is used to indicate that the next line in the text should be a part of the first line.

### 4.2.1 Map request

Like everything in RDF it is identified by an URI, so the URI of the map would be the only argument necessary for this request. If it is not provided the server could choose to either send an error message (as described in 4.2.11) or to send an arbitrary map, perhaps some kind of default map. If the client knows of the container the map is included in it can be the optional second argument. To identify this request it starts with the codeword GETMAP, followed by the arguments. Example:

```
GETMAP http://www.example.org/examplemap/\
http://www.example.org/examplecontainer.rdf LCP/0.1
```

### 4.2.2 Response to a Map request

It is assumed that when the client makes a maprequest it wants to display the map. The information about it is divided into 3 parts, the general information about the map, information about the concepts and last is the information about the concept-relations. All these three parts come in the same order in the protocol and are separated by two new lines. For every part abstract, graphical, navigational and content information can be included, which was the important parts of the context-map according to section 3.2. To separate them each type of information are inside an angle bracket pair. Example of the structure:

```
<[Abstract]><[Graphical]><[Navigational]><[Content]>
```

The general information about the map consists of the abstract and the graphical information in this version, but if there are following angle bracket pair it is reserved for the navigational and content information. The abstract information is the URI and title of the map. The graphical information is the height and the width of the map in pixels. Example:

```
<http://example.org/examplemap;Example map><75,75>
```

The second part, about the concepts, have an angle bracket pair for all 4 types in the following order:

- abstract information is the URI of the concept, a title and a description. Example:

  ```
  <http://example.org/exampleconcept;Concept example;Example\
  of a concept>
  ```

- The graphical information is the location of the bounding box where the concept should be placed on the map, the size of it and the style of the concept. Example:

  ```
  <35,35;12,10;rectangle>
  ```

- Navigational information is the detailed map and a list of the contextual neighbourhood of the concept. Example:

  ```
  <http://www.example.org/examplemap2;http://www.example.org/\
  map1,http://www.example.org/map2>
  ```

- content information is a list with every content assigned to the concept as its URL and the MIME-type. Example:

  ```
  <http://www.example.org/pic1.jpg,Image/jpeg;\
  http://www.example.org/ex.html, text/html>
  ```

If the map does not contains any concept an empty pair of angle brackets are sent instead.

The concept-relations refer to the concepts by their number in the order that they are in the protocol and not by their URI. So the order of the concepts in the protocol sent are important to remember for a client. The structure of information for the concept-relations are basically the same and the content and navigational information is structured exactly same way for the concept-relations as for the concepts. The rest of the information for the concept-relations are structured in the following way:

- Abstract information is the URI of the reification followed by the URI of the property. This is followed by the number of the subject concept and the number of the object concept of the statement. Last is the title and the description. Example:

  ```
  <http://www.example.org/exreification;\
  http://www.example.org/exProperty;1,2;example-Statement,\
  example of a statement>
  ```

- Graphical information of a concept-relation are the points that the line is connected through. It is actually just a list of numbers but should be paired. The first number is the x-coordinate and the second the y-coordinate and so on. A location and size of a bounding box for the title follows and then how the line and arrow should be styled. Last is the direction the arrow, if it should be in a forward or backward direction to the line. With an *f* it is indicated that the arrow should be at the end of the line, which is the last point given in the points for the line. A *b* indicates a backward direction. Example:

  ```
  <6,7,4,5,8,9;11,22,10,10;dotted,fullarrow,f>
  ```

If the map does not contains any concept an empty pair of angle brackets are sent instead. An example of a full map is included in Appendix C.

### 4.2.3 Request of a contextual neighbourhood

To request a contextual neighbourhood means that all the maps where a certain concept occurs is requested. So this would again mean that the URI of the concept would be enough. Usually a contextual neighbourhood is requested when a concept have been found in a context-map. To exclude that map in the response, the URI of the map can be added as an argument. The word for identifying this request is GETNEIGHBORHOOD. Example:

```
GETNEIGHBORHOOD http://www.example.org/exampleURI/\
http://www.example.org/examplemap/ LCP/0.1
```

### 4.2.4 Response to a contextual neighbourhood request

This response should basically consist of the set of maps that make out the contextual neighbourhood. Depending on the request this set could look a little bit different. If the request only has the the first argument, the URI of the concept, the response will be a list of all the context-maps containing that concept. If the request has a second argument which is the map where the concept was found, all maps except that map should be returned. That is actually a punctured set of a contextual neighbourhood that is returned. The structure of this response starts with the URI of the concept inside an angle bracket pair, followed by two new lines and then the list of context-maps follows and one map is on one line and inside an angle bracket pair with the URI of the map and the title of the map or a short information about that map. If no contextual neighbourhood can be found an empty pair of angle brackets are sent instead. Example:

```
LCP/0.1 GETNEIGHBOURHOOD

<http://example.org/exampleURI/>

<http://example.org/mapURIone/;Map one>
<http://example.org/mapURItwo/;Map two>
<http://example.org/mapURIthree/;Map three>
```

When requesting a context-map the contextual neighbourhood is included for every concept and concept-relation. That would mean that a contextual neighbourhood would not have to be a possible request, since it is already provided. The reason for still having it is that a concept might be found somewhere else than inside a context-map. With this request it is possible to get the contextual neighbourhood of an arbitrary concept. Another reason that could happen was that the whole contextual neighbourhood for a concept is not provided in the context-map response to save bandwidth. Then this request would be a complement to the maprequest.

### 4.2.5 Metadata request

The only argument would here as well be the URI for the concept the client would like metadata about. But all information about one resource could be quite extensive to make use of, so there are two ways to reduce the metadata information loaded. One is to list all the properties of interest for that concept, or to only include the metadata that is included in a certain RDF-vocabulary. The code word GETMETADATA is used as identifier and below is an example with the list of properties as the second argument. The entries in this list are separated by a comma. Example:

```
GETMETADATA http://www.example.org/exampleURI/ \
http://www.example.org/title/, \
http://www.example.org/description/ LCP/0.1
```

### 4.2.6 Response to a Metadata request

The request of metadata could be done in several ways, but the structure of the response is the same. When requesting metadata it is basically a request of which statements have this concept as the subject. The structure starts with the URI of the concept followed by two new lines and then the rest of the statement follows each on every line and every entry is inside an angle bracket pair. The object is sometimes a string literal and sometimes a URI. Example:

```
LCP/0.1 GETMETADATA

<http://example.org/exampleURI>

<http://example.org/exPredicate1/;http://example.org/\
exampleObject/>
<http://example.org/exPredicate2/;Example string 1>
<http://example.org/exPredicate3/;Example string 2>
```

### 4.2.7 Content request

A content component is identified with the URL, which also tells where the content component is located. Apart from that the MIME-type is included to tell what kind of file the proxy server should download and then retransmit. This is a security issue, since it is important to deny files of a certain MIME-types to be downloaded if they could infect the client or the proxy server. The word for identifying this request is GETCONTENT. Example:

```
GETCONTENT http://www.example.org/index.html text/html LCP/0.1
```

### 4.2.8 Response to a Content request

This response deviates from the principle of that one request should equal one response and a waiting state for the client is created. When a request of content is made it might take a while to get the content to the server, therefore a response is sent to the client stating whether the server accepts to download the content or not. This response consists of the URL of the content and an integer stating the status, all this is inside an angle bracket pair. Currently the status 1 equals accepted and the status 0 not accepted. Example:

```
LCP/0.1 GETCONTENT

<http://www.example.org/expic.jpg;1>
```

With the status 0, the content is not accepted to be downloaded to the server. If the status is 1 the content will be downloaded to the server and the client could expect the server to tell it when this download is finished, so the connection between the client and the server has to be kept alive. This is done with a message with the URL on the server the content is located together with the original URL. Example:

```
LCP/0.1 GETCONTENT

<http://www.example.org/expic.jpg;\
http://www.exampleserver.org/expic.jpg>
```

If the download somehow fails an error response described below is sent instead.

### 4.2.9 Content request of a concept

A set of content components could be assigned to a concept. This is a request for a list of the content components assigned to a concept. To request it the URI is the primary argument. The optional second arguments would be the MIME-types the content in the response should have. If the client only wants to know of content components in the form of html it could be specified. Without the second argument MIME-types will not be taken into consideration. The codeword for identifying this request is GETCONCEPTCONTENT and below follows an example:

```
GETCONCEPTCONTENT http://www.example.org/exampleconcept\
text/html,Image/jpeg
```

### 4.2.10 Response to a content request of a concept

The structure of this response is the same for every request, but if MIME-types are specified in the request only the content components of that type would be included. The response includes the concept and after that the list of content components are listed with its URL, MIME-type and a title. The structure is the same as for the contextual neighbourhood, but with the MIME-type. If no content is found a pair of empty angle brackets are sent. Example:

```
LCP/0.1 GETCONCEPTCONTENT

<http://www.example.org/exampleconcept>

<http://www.example.org/;text/html;The example homepage>
<http://www.example.org/expic.jpg;Image/jpeg;Example picture>
```

This list is like the list of a contextual neighbourhood included for each concept in the context-map response. As in the case of the contextual neighbourhood all concept might not be discovered by maps and the ability to see a list of the assigned content component is provided. It could here as well be

the case that all content components are not in the context-map response, if so this request could be used.

### 4.2.11 Error response

This response is not fully developed since the main focus is to design the responses when things works out. Though, some kind of error message should be sent if a failure or some sort of error occurs. The response is the request inside an angle bracket pair together with the word FAILED. Example:

```
LCP/0.1 ERROR

<GETMAP http://example.org/MapURI;FAILED>
```

This response could of course be more detailed and tell the client what was wrong, if the syntax of the request was incorrect or if it just failed to process it. Though this will hopefully be done in a future version.

## 4.3 Choice of syntax

The design of the protocol was partly inspired by the Hypertext Transfer Protocol (HTTP), especially the request part of it. Mainly because it is a simple protocol. As HTTP uses GET, POST and so on, this protocol uses GETMETADATA, GETMAP and so on to identify the request. The first argument for the request is always the URI of a resource and, except for the content request, the second argument is optional. With this second optional argument a list of several URIs can be specified. They are separated by a comma as in the example of the metadata request. The request always ends with LCP/ followed by the version number (0.1 is the current version) of the protocol and a new line. After that some properties about the client could be specified the same way as it is done in HTTP. And as in that protocol the end is marked with two new lines.

When designing the responses to these requests the main focus was to make the map response as good as possible regarding compactness without loosing too much information. It was designed to include the least amount of information for a thin client to display the maps. So what is in the design is considered to be the least amount of information needed. In future version it can be possible to extend the graphical, abstract, content or navigational information since they are inside an angle bracket pair, if the information that needs to be added are in one of those categories. One thing not directly taken into consideration is the styling information of certain concept and concept-relations. In this version they are just handled by a text stating how certain things should be displayed inside the graphical information.

This could be developed further and perhaps be an own category, hence a new angle bracket pair should be added to include this information. If no information is found for a category the angle bracket pair should be left empty to state that no information was found.

The angle brackets are used to separate the different parts, but they have another important function. Every concept and concept-relation use one line for the information about it, but sometimes the title (or something else) uses a linebreak. To recognize if the linebreak mean that a new concept is coming or if it was just a linebreak it can be detected if they were inside or outside the angle bracket pair. Inside those the different parts are sometimes separated by a comma (,) and sometimes by a semicolon (;). The reason for this is that the comma is used for a list or information that is closely connected somehow. The semicolon is used otherwise. A problem might come up that commas and semicolons are used by for example a description of a concept. Then they should have a backslash(\) included before so that the client understand if it is a separator or just a part of the text. A way of reusing information in the protocol is to use the ordering of the concepts for the information about the statement. Instead of using the whole URI, the number of the concept is used instead. It preserves parts of the layout layer discussed in 3.2.1 since otherwise both the Concept-URI and its Layout-URI would have to be included to be certain to what concept it is otherwise one concept could only appear once in one map.

The syntax of the neighbourhood and metadata response is just mainly a list of properties for a certain resource. The semicolon (;) is used for separating on item in those list and they are inside an angle bracket pair. The angle brackets are used to show a resemblance with the rest of the responses and that a new line could be inside the title of the map for the neighborhood response or in the strings for the metadata request. For the content response the angle brackets are used just to use a similar syntax. A rough description of the syntax in EBNF is included in Appendix B.

# 5 Proxy server for a Concept Browser

One of the goals of this thesis project was to create a proxy server for gathering information to a Concept Browser running on a client that have a limited capacity regarding memory and bandwidth. The basic problem lies in the structure of the information when using RDF. So the idea is that when the client needs information it requests the proxy server. The request is processed by the server which sorts out and creates a reply containing necessary information for the client composed according to the protocol described in chapter 4. Possible clients are:

- An unsigned Java Applet running on a webpage. It has a restriction of only being allowed to gather information from the same source as it was loaded from. That would be possible with a proxy server.
- A Flash program running on a webpage.
- Portable machines, such as PDA's and mobile phones with a lower bandwidth and memory capacity.

## 5.1 Request handling

The idea of a proxy server is to be a middle hand in the process of transmitting a request and a response between the client and the server. If a client requests something from a proxy server it gets an answer from the proxy server even though the information is (usually) located on another server. The proxy server finds the requested information and downloads it. Some processing of the information is probably needed and then the information is sent to the client. To keep the request and the response between the client and the proxy server easy to handle, the request would be handled like this:

1. Receive and identify the type of request.
2. Harvest information and fill cache.
3. Extract the necessary information.
4. Compose and send a reply.

Another possible way could be to not await all information, but to send parts of it directly when it is found. This would mean that we keep a lot of states in the proxy server and that is not wanted since it could lead to a tricky matchmaking of what information has been sent or not. So to keep it simple, one request equals one response. This goes hand in hand with the protocol defined for transmitting the information and was one of the reason to why the protocol is stateless.

### 5.1.1 The connection between the client and the proxy server

The proxy server needs to receive the request, which should be done by some sort of connection is opened between the client to the server. Since this server will serve more than one client it is possible that several requests might come to the server at the same time. Therefore the server should be able to start processing a request even if it is processing another, which means that the server has to be multithreaded. When this connection is made it should remain open at least until a response have been sent in that connection. Otherwise it would be impossible for the client to remember what request corresponds to what response as mentioned in chapter 4.

When the request is received it will be parsed and identified. The possible request were described in the previous chapter.

## 5.2 Techniques for harvesting information

When the request is identified necessary information often needs to be gathered. Since information is (probably) not located on the proxy server it needs to be downloaded from relevant servers, that first has to be located.

Now two possible scenarios could appear, either the location of the information is known or it is not known. Let us now consider the first case. This could either be the case that the information is already on the proxy server through some sort of caching and it could be easily referenced. It could as well be the case that the information is not in the cache of the proxy server and needs to be downloaded to the server. The location of the information can in most cases be discovered through the property *CVN:IncludeContainer*.

However, when harvesting information only through downloading containers the set of available information gets limited to the information in those containers. This will show when information should be extracted, since all statements where the resource is the subject are usually placed in the same container. On the other hand, when looking for statements where a certain resource is the object is much harder since those statements are usually spread out in several containers.

It is more troublesome if the location of the information is not known. How things could be found on the Semantic Web through Conzilla are described in [2]. It is done by sending a question out on the Edutella network, which is a peer to peer network using RDF. Question could be asked by using the Edutella Query Language QEL [1] on that network. Another technique using RDF query language (RDQL) is the Sesame project described in [5].

Through this technique it is possible to extract only the information needed from containers stored on that system and don´t have to download the rest of it. That would save memory and probably some time for the proxy server, but to use it the location would probably have to be known. Edutella and Sesame are two system that could help provide information for the proxy server. The main difference when for the proxy server to use them is that Edutella is built as a peer to peer application where Sesame is more built as a database/server on the internet. These two techniques could be used even when the location of information is known. That way only the information needed would be downloaded and not the whole container and when these techniques are fully developed they are probably to be preferred.

## 5.3 Caching techniques

When the information is downloaded to the proxy server it should at least be kept on the proxy server until the information needed has been extracted and a response has been made. After that the information should be kept in a cache on the proxy server, since the information used to create a response for one request is usually closely related to the next request and that response could be created quicker. Deciding exactly what to cache and for how long is a tricky question. One important part of information good to cache is the statements that includes information on where to find more information about certain resources, like the property *CVN:IncludeContainer*. If that points to a container it should perhaps not be downloaded but it is important information for the proxy server. If they were to be downloaded directly once they were found would lead to a quick overflow of the cache and since the downloaded containers in turn could include links to other containers it would necessary to download them as well. It would be necessary to decide when to stop so not too many containers is downloaded and the best would be to not download the containers until they were really needed. This way memory is saved, but it relies on that things could be quickly downloaded to the proxy server.

Apart saving the information where more information is located some of the statements downloaded should be kept in the cache. To save memory a limit for the cache is needed to not overflow it. The tricky question is when to drop information from the cache. It is important to keep statements that are often referenced in the cache and drop the statements used less often. This could be solved by making the cache a First in First out queue. All statements are put in the queue once they arrive to the proxy server. If the cache starts to get full it would cut of at the end of the queue which are the statements less frequently used. To keep the information frequently used at beginning of the queue it is updated every time a statement is referenced by putting it in the beginning of the queue again. This way the

statements often used are kept in the cache and the ones less frequently used are dropped. The cache of information should be separated from the cache that indicates where more information is located. That is because those statement might not be used that much and be dropped by the cache even if they keep valuable information. If two queues are used this would be avoided.

## 5.4 Extract the necessary information

When the information wanted is harvested it should be possible to extract necessary information to be able to create a response to the request. It could be the case that some information links to other information, for example the property *CVN:includeContainer*. If so, the harvesting step should be repeated. When everything needed is loaded and cached required information can be extracted from the graph of statements.

When extracting information several situation can occur where different parts of the statements are known. Three common situations for the proxy server are:

- The subject and the property of the statement are known and what is searched for is the object.
- Only the subject is known and all those statements with this resource should be extracted.
- The property and the object are known and the subject should be extracted.

### 5.4.1 Extracting metadata

Metadata for a concept is extracted by finding the statements where the concept-resource is the subject. Metadata can be extracted in two ways, either the predicate for that resource is known. In that case the statements with the concept-resource as the subject and all the predicates should be extracted. If the predicate is not known all statements where the concept-resource is the subject are extracted.

### 5.4.2 Extracting a contextual neighbourhood

A contextual neighbourhood of a concept is the context-maps that contain that concept. This means that a search has to be done in the reverse way to the metadata-search, since the maps are referring to the concept. This means that the predicate and the object in the RDF-statement are known, but not the subject.

If a concept-resource is included in a map when using the Conzilla vocabulary it is the value of the property *CVL:displayResource* as in figure 7. The subject of such a property is the layout of a concept which can be included in a context-map. The next thing to do is to find if this layout is included in a map. The maps found are the contextual neighbourhood of the concept.

### 5.4.3 Extracting a context-map

Extracting a context-map is done by searching as in section 5.4.1. Here all the properties are predefined and known in advance. It is necessary to know what kind of RDF vocabulary that is used and what properties it defines for a context-map. For this thesis the Conzilla vocabulary is used and a part of it not mentioned before is the styling part of it. It has the URI *http://kmr.nada.kth.se/rdf/style#* and will be abbrieviated *CVStyle:*. Another vocabulary called Dublin Core is used as well, and the URI *http://-purl.org/dc/elements/1.1/* is abbrieviated *DC:*

For the Conzilla vocabulary a context-map has the type *CVL:ConceptMap*. That resource has the graphical information, which is the size of the map, through the property *CVL:dimension*. The title is found through the property *DC:title*. The maps consists of several layouts for concepts and Statements. From the concept layout the information about the concept can found, according to the following:

- ***Abstract information*** is basically the metadata about the actual concept on the abstract layer. To find it the property *CVL:display-Resource* is provided which has the value of the actual concept. The concept has an URI, and the title and description can be found through the properties *DC:title* and *DC:description* for that URI.
- ***Graphical information*** is found through the properties *CVL:location* and *CVL:dimension*. To know how the concept should be styled can be found by the property *CVStyle:boxStyle*
- ***Navigational information*** is provided through the property *CVN:-hyperlink* and the contextual neighbourhood is found as described in section 5.4.2.
- ***Content information*** How this should be handled is not yet defined in the Conzilla vocabulary, so currently this is not possible.

The statement layout gives information about a statement according to the following:

- ***Abstract information*** Is found the same way as from the concept layout, through the property *CVL:displayResource*, which indicates which reification it is displaying. It links to the abstract information through the properties *DC:title* and *DC:description*.
- ***Graphical information*** Is found through the properties *CVL:statementLine* and *CVStyle:lineStyle* and for the arrow the properties *CVStyle:lineHeadStyle* and *CVStyle:lineHeadInLineEnd* is used.
- ***Navigational information*** and ***content information*** Are treated the same way as for the concept layout

## 5.5 Compose a response and send it

This part goes hand in hand with the previous step, since they could be done at the same time. Exactly how to compose the response was described in chapter 4. This is actually done by combining a string with the information that is extracted.

## 5.6 Implementation

To implement a proxy server that met the requirements an object oriented design was done that ended up with the class diagram in figure 8. The main goal of the design except meeting the requirements was to keep a simple design without loosing possibility to extend it. An implementation has been done according to this design. How each requirement is met is discussed in the following chapter.

### 5.6.1 External API:s used

The Conzilla application provides several useful classes for loading and locating context-maps represented in RDF with the Conzilla Vocabulary. A class diagram over the important classes are shown in figure 9. These classes have formed a base for building the application in this thesis project. This way a resemblance with the Conzilla project is made. Since they are written to load context-maps made in RDF they are very useful for the purpose of this thesis.

### 5.6.2 Connection

The connection for sending the requests and responses is established through HTTP. The reason for this was that the J2ME on the mobilephone, the client the server was primarily created for, could provide a HTTP-connection. The connection will listen and receive connections through some TCP-port,

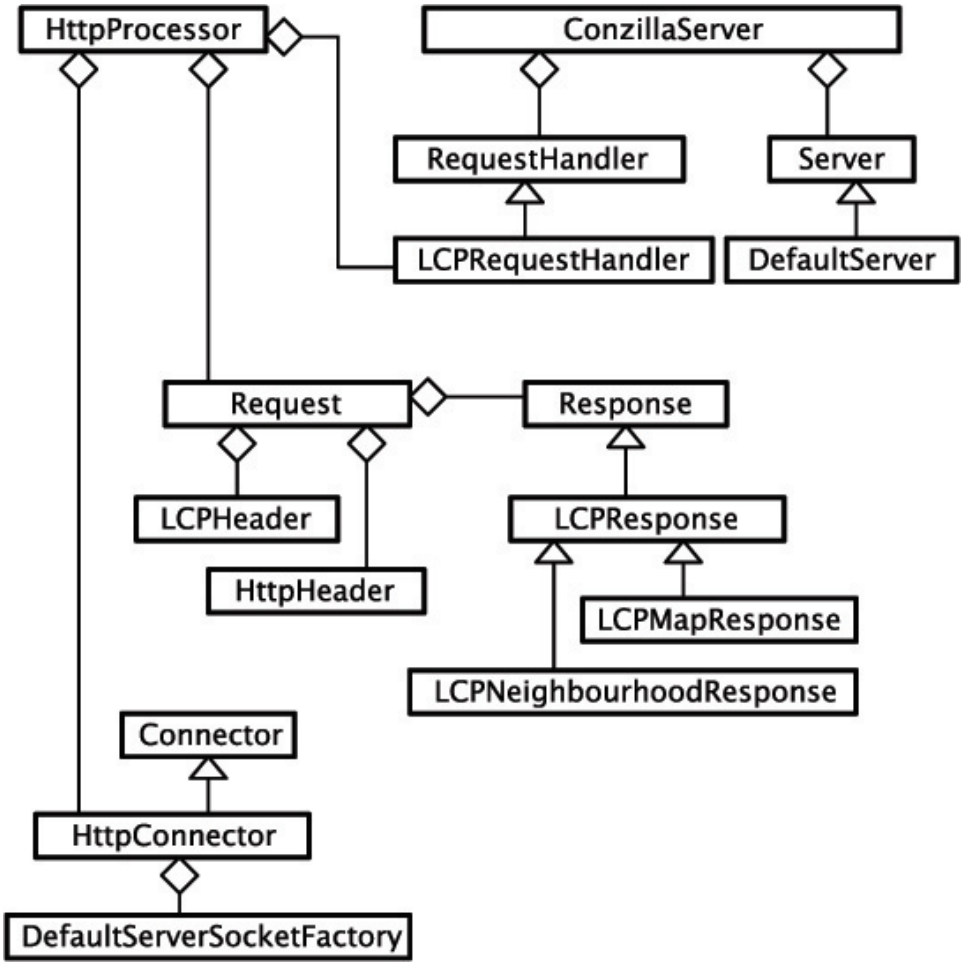


Figure 8: Class diagram of the Conzilla classes used in this project.

currently 8082. For this class to handle several requests and process them concurrently a request is assigned to be handled by its own thread. These threads are instances of the class *HttpProcessor* which handles the HTTP part of the communication. The class parses the protocol and the content of it which is the LCP-protocol. This information goes into instances of the classes *HttpHeader* and *LCPheader*. These are given to the *LCPRequestHandler* which identifies what kind of request this was and creates an instance of a *Response*. Exactly what instance depends on the request.

### 5.6.3 Harvesting and storing information

To be able to load and store information on the proxy server classes from the Conzilla project have been used. The important one is the *ResourceStore*

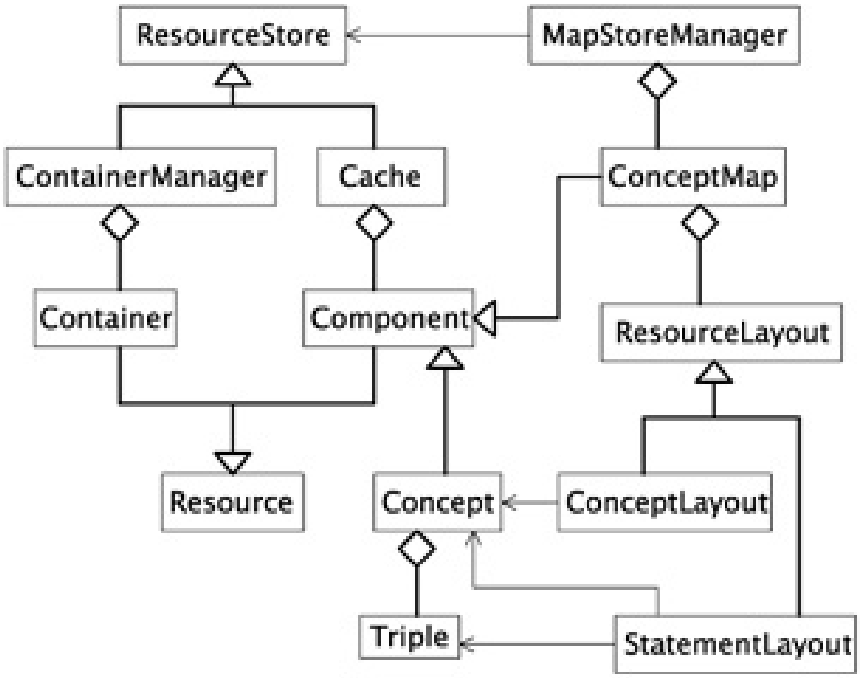


Figure 9: Class diagram of the Conzilla classes used in this project.

which handles all information of a downloaded container. It has the possibility to load a container into memory and from it information can be retrieved. A static instance of that class is created inside the *LCPRequestHandler*. So when a container needs to be loaded it is done through this instance. It is used for extracting information as well and it can be seen as the information centre. The information is cached inside the *Cache*, which is the cache of the proxy server. The current implementation is not very flexible and it stores everything from a loaded container. This way information will only have to be downloaded once, but this will eventually lead to an overflow of the cache. The current solution to locate more information relies on that the property *CVN:includeContainer* is provided. If this property is not there and the information does not exist in the cache it cannot be found.

### 5.6.4 Extracting information and create a response

This is done inside the different subclasses of *Response*. The name reveals what kind of service is done. For example if a map was requested, an instance of the class *LCPMapResponse* is created and is responsible for the response. If metadata was requested an instance of the class *LCPMetadataResponse* is

created, and so on. These instances should reference the cached information. If it does not exists there information need to be harvested.

To extract information about a map could is done this way, but since the maps are an important part of the Concept Browser, special classes are created to handle them. The class *MapStoreManager* is used, which is initiated by a *ResourceStore* and the URI of the map. When it is initiated an instance of the class *ConceptMap* is created which holds information about the map. From that the information about the concepts and concept-relations can be retrieved through instances of *StatementLayout* and *ConceptLayout*. They are classes to represent the Statement layout and Concept layout discussed in section 3.2.1.

### 5.6.5 Environments

The chosen programming language for the project is Java. The reason for this choice was:

- The code for Conzilla is written in Java and therefore a natural choice for this code as well
- by using the same code the Conzilla classes can be used in the code for this project since Java is an object oriented programming language.
- The external Jena API is written in Java, which could be used in an extension of this code. The Conzilla code makes use of the Jena API.
- The programs written in Java is compiled to run on the Java Virtual Machine, so the compiled program can run on platforms other then they are compiled on. The virtual machine also provides an automatic garbage collection.

To compile the Open source program Ant has been used. It is a program designed to compile programs written in Java, where the instructions are defined in an XML-file. Such a file already existed for the Conzilla project, so that file could easily be extended to compile the server.

# 6 Conclusions

The main problem for a Concept Browser as a thin client is to handle the vast amount of information. This was the reason that this project was initiated and to create a serverside solution the questions in the introduction needed to be answered.

## 6.1 The requests

The first question I posed was: *What could be requested of a proxy server to enable the idea of a Concept Browser as a thin client?* The most essential part for a Concept Browser are the context-map and the referencing concept and concept-relations that belongs to it. These essentials are all in one request. Apart from that it is necessary to navigate and explore contexts and concepts. Therefore the requests for a contextual neighbourhood for a concept to navigate and a list of the content-components assigned to it for exploring are needed. To request those content-component of the proxy server could help certain types of clients and should therefore be possible as well. Another important request is the metadata request since a concept or a content-component can have metadata not retrievable by in other ways. All in all I have covered the needs of a Concept Browser in five requests.

## 6.2 The protocol

When the possible requests had been recognized a way of communicating a request and its response needed to be designed. The second question I posed was therefore: *What kind of protocol would be needed and how should it be designed?* The protocol designed was text based and stateless and is running over HTTP. By doing that a compromise could be made between not using too much bandwidth or memory and CPU time on the client side. The protocol is designed to be simple and compact. The information sent is rather limited, though, by the angle brackets it can be extended.

## 6.3 Harvesting and caching

The last question I posed was: *For these purposes, how should information from the Semantic Web be harvested and cached?* How to harvest the Semantic Web is a tricky question and a problem not yet totally solved for the Semantic Web, but a solution that work for the most common scenarios for the proxy server relies on the property *CV:includeContainer* to point out where information about a resource can be found. As long as that information is provided the proxy server can find information even if it cannot be guaranteed that all relevant information is found.

The information and the information where the information resides, should be kept in two different caches. To cache information is a way to optimise performance by not having to download information to the proxy server too often. The cache that is used in the implementation is currently just caching everything that comes into the proxy server. This could of course lead to an overflow, but if the set of information is limited it will work for a first version of the proxy server.

## 6.4 Future perspectives

The protocol was designed for the requests that were needed for a Concept Browser, but the response part could be improved in future versions. The response to a maprequest have graphical information to draw a map, but it does not say exactly how. For example if a line should be dotted or if a box should have sharp or rounded corners. For instance some kind of stylesheet could be added to the protocol instead of the current solution of including it in the graphical information. Another related thing that could be improved is to put an arrow at both ends of a line, but it is not possible in this version and it is not supported yet in the Conzilla vocabulary. The request of a content component is perhaps not the task for the proxy server, but could help certain clients. Unfortunately that request makes a state appear in the proxy server which is against the design goal put up for the protocol. If the protocol remains stateless and this feature is kept this needs to be addressed. The error response needs to be further developed, for example with error codes.

The KMR group at CID is taking part in the Edutella project and in the Master's thesis by Henrik Eriksson [2] a possible way to ask questions to the network from Conzilla was proposed. To use the Edutella network would mean that a larger amount of information would retrievable, but the set of information is limited to the one on that network and exactly how to use Edutella for harvesting remains to be investigated.

# References


[1] *Edutella query language, QEL.* http://edutella.jxta.org/spec/qel.html. Last visited: January 31, 2006.

[2] Henrik Eriksson. *Query Management for The Semantic Web.* Master's thesis, Uppsala University, March 2003.

[3] *Hypertext Transfer Protocol, HTTP/1.1.* http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf. Last visited: January 31, 2006.

[4] *Java micro edition.* http://java.sun.com/j2me/index.jsp. Last visited: January 31, 2006.

[5] Arjohn Kampman Jeen Broekstra and Frank van Harmelen. *Sesame, An Architecture for Storing and Querying RDF Data and Schema Information*, chapter 7, pages 197–222. The MIT press, 2003.

[6] *Knowledge Management Research Group.* http://kmr.nada.kth.se. Last visited: January 31, 2006.

[7] Ambjörn Naeve. *Conceptual Navigation and Multiple Scale Narration in a Knowledge Manifold.* CID-52, TRITA-NA-D9910, Dept. of Numerical Analysis and Computer Science, KTH, Stockholm, Sweden, 1999.

[8] Ambjörn Naeve. The concept browser, a new form of knowledge management tool. In *Proceedings of the 2:nd European Web-Based Learning Environments Conference (WBLE 2001), Lund, Sweden*, 2001.

[9] Mikael Nilsson and Mattias Palmér. *Conzilla, Towards a Concept Browser.* CID-53, TRITA-NA-D9911, Dept.of Numerical Analysis and Computer Science, KTH, Stockholm, Sweden, 1999.

[10] *RDF Primer.* http://www.w3.org/TR/rdf-primer/. Last visited: January 31, 2006.

[11] *RDF Vocabulary Description Language 1.0: RDF Schema.* http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. Last visited: January 31, 2006.

[12] *RDF/XML Syntax Specification.* http://www.w3.org/TR/rdf-syntax-grammar/. Last visited: January 31, 2006.

[13] *The Semantic Web.* http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2. Last visited: January 31, 2006.

# Appendices

## A Conzilla Vocabulary

This Appendix includes parts of the vocabularies for Conzilla.

### Layout

The following resources and properties are used for the layout of the context-maps, concepts and concept-relations. The whole URI is not typed, but it can retrieved by adding *http://kmr.nada.kth.se/rdf/graphic#*.

Resource *ContextMap*
Resource *NodeLayout*
Resource *ConceptLayout*
Resource *StatementLayout*
Resource *LiteralStatementLayout*
Property *subLayerOf*
Property *displayResource*
Property *location*
Property *dimension*
Property *literalLocation*
Property *literalDimension*
Property *bodyVisible*
Property *subjectLayout*
Property *objectLayout*

Property *statementLine*
Property *statementLinePathType*
Property *boxLine*
Property *boxLinePathType*
Resource *LinePathType_Straight*
Resource *LinePathType_Curve*

Property *horizontalTextAnchor*
Property *verticalTextAnchor*
Resource *Center*
Resource *North*
Resource *South*
Resource *East*
Resource *West*

### Navigation

The following properties are used for navigation purposes. The whole URI is not typed, but it can retrieved by adding *http://kmr.nada.kth.se/rdf/-navigation#*.

Property *hyperlink*
Property *includeContainer*

### Style

The following properties are used for styling the context-maps, concepts and concept-relations. The whole URI is not typed, but it can retrieved by adding *http://kmr.nada.kth.se/rdf/style#*

Property *styleInstance*
Property *styleClass*
Property *boxStyle*
Property *boxFilled*
Property *boxBorderStyle*
Property *boxBorderThickness*
Property *lineStyle*
Property *lineThickness*
Property *lineHeadInLineEnd*
Property *lineHeadStyle*
Property *lineHeadFilled*
Property *lineHeadWidth*
Property *lineHeadLength*
Property *lineHeadLineThickness*
Property *boxLineStyle*
Property *boxLineThickness*

# B EBNF for LCP

```
(* This describes version 0.1 of LCP in EBNF. The protocol
   consists of two different parts, the requests and the
   responses. *)

Request = MapRequest | NeighbourhoodRequest |
          MetadataRequest | ContentRequest |
          ConceptContentRequest;

Response = MapResponse | NeighbourhoodResponse |
           MetadataResponse | ContentResponse1 |
           ContentResponse2 | ConceptContentResponse |
           ErrorResponse;

MapRequest = 'GETMAP ',URI,' ',[URI,' '],'LCP/0.1',
             2 * new line;

MapResponse = 'LCP/0.1 GETMAP',2 * new line,
              '<',URI,';'desc-string,'><',Integer pair,'>',
              2*new line,
              (('<>',new line) |
              (MapConcept,{MapConcept})),new line,
              (('<>',new line) |
              (MapRelation,{MapRelation}),new line;

NeighbourhoodRequest = 'GETNEIGHBOURHOOD ',URI,' ',[URI,' '],
                       'LCP/0.1',2*new line;

NeighbourhoodResponse = 'LCP/0.1 GETNEIGHBOURHOOD',
                        2 * new line,'<',URI,'>',2 * new line,
                        (('<>',2*new line)|
                        ('<',URI,';',desc-string,'>',new line
                        {'<',URI,';', desc-string,'>',
                        new line},new line));

MetadataRequest =  'GETMETADATA ',URI,' ',[{URI,','},URI,' '],
                   'LCP/0.1',2 * new line;

MetadataResponse = 'LCP/0.1 GETMETADATA',2 * new line,
                   '<',URI,'>',2 * new line,
                   (('<>',2*new line)|
                   ('<',URI,';' (desc-string | URI) ,'>',
                   new line,
```

```
                  {'<',URI,';' desc-string | URI ,'>',
                  new line}, new line));

ContentRequest = 'GETCONTENT ',URI,' ',MimeType,' ',
                 'LCP/0.1',2 * new line;

ContentResponse1 = 'LCP/0.1 GETCONTENT',2* new line,
                   '<',URI,';',('0'|'1'),'>',2*new line;

ContentResponse2 = 'LCP/0.1 GETCONTENT',2*new line,
                   '<',URI,';',URI,'>',2*new line;

ConceptContentRequest = 'GETCONCEPTCONTENT ',URI, ' ',
                        [{MimeType,','},MimeType],2*new line;

ConceptContentResponse = 'LCP/0.1 GETCONCEPTCONTENT ',
                         2 * new line,'<',URI,'>',
                         2 * new line,
                         (('<>',2*new line)|
                         ('<',URI,';', MimeType,
                         ';',terminal string,'>', new line,
                         {'<',URI,';', MimeType,
                         ';',terminal string,'>', new line},
                         new line));

ErrorResponse = 'LCP/0.1 ERROR',2*new line,
                '<',('GETMAP '|'GETNEIGHBOURHOOD '|
                'GETMETADATA '|'GETCONTENT '|
                'GETCONCEPTCONTENT '),URI,';FAILED>',
                2*new line;

MapConcept = '<',URI,';',desc-string,';',desc-string,'><',
             Integer pair,';',Integer
             pair,';',box type,'><',[URI],';',[{URI,','},
             URI],'><',[{URI,',',mimetype,';'},URI,',',
             mimetype],'>',new line;

MapRelation = '<',URI,';',URI,';',Integer pair,';'terminal
              string,';',desc-string,'><',Integer pair,
              ',',Integer pair,{','Integer pair,',',
              Integer pair},';',line type,',',arrow type,',',
              direction,'><',[URI],';',[{URI,','},URI],'><',
              [{URI,',',mimetype,';'},URI,',',mimetype],'>',
              new line;
```

```
URI = (*As defined in RFC 2396 at http://www.ietf.org/ *)

digit = '0' | '1' |'2' | '3' | '4' | '5' | '6' |
        '7' | '8' | '9';

new line = ? ISO 6429 character Carriage Return ?,
          ? ISO 6429 character Line Feed ?

MimeType = (* As defined in RFC:s 2045-2049 at
              http://www.ietf.org/*)

desc-string = {(terminal character - ';')-',','|\;|\,};
              (* a terminal character as defined in the
                 EBNF-standard*);

Integer pair = digit,{digit},',',digit,{digit};

line type = ('continous'|'dotted'|'dashed'|'dashdot'|
             'dashdotdot'|'dashdotdotdot')

arrow type = ('arrow'|'varrow'|'sharparrow'|
              'bluntarrow'|'diamond'|'none')

box type =('rectangle'|'roundrectangle'|'diamond'|'ellipse'|
           'flathexagon'|'upperfive'|'lowerfive'|'invisible')
```

# C Example of a map in LCP

In the following example of a context-map in LCP a \ will be used to break the line to fit the margins and it is not a part of LCP. This example corresponds to the map in figure 1.

```
LCP/0.1 GETMAP

<http://www.conzilla.org/rdf/demo;\
http://www.conzilla.org/rdf/demo><185;203>

<http://www.conzilla.org/rdf/concepts#16925b0f7ee40f955;\
Maps;Maps><48,12,72,24;rectangle><;><http://www.google.com/,\
text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee410070;\
Food;Food><6,72,42,24;rectangle>\
<http://www.conzilla.org/rdf/demo#mat;>\
<http://www.google.com/,text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee4112e9;\
Course;Course><114,72,54,24;rectangle><;>\
<http://www.google.com/,text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee4117c2;\
Culture;Culture><96,108,54,24;rectangle><;>\
<http://www.google.com/,text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee411d39;\
Business;Business><6,108,60,24;rectangle><;>\
<http://www.google.com/,text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee4120db;\
Entertainment;Entertainment><36,144,96,24;rectangle><;>\
<http://www.google.com/,text/html>

<http://www.conzilla.org/rdf/concepts#16925b0f7ee45b78b;\
http://www.w3.org/2000/01/rdf-schema#subClassOf;1,0;\
concepts#16925b0f7ee45b78b;concepts#16925b0f7ee45b78b>\
<84,36,84,72,48,78;0,0,0,0;continous,arrow;f><;>\
<http://www.google.com/,text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee45c055;\
http://www.w3.org/2000/01/rdf-schema#subClassOf;4,0;\
concepts#16925b0f7ee45c055;concepts#16925b0f7ee45c055>\
<84,36,84,72,60,108;0,0,0,0;continous,arrow;f>\
<;><http://www.google.com/,text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee45cc0d;\
http://www.w3.org/2000/01/rdf-schema#subClassOf;5,0;\
```

```
concepts#16925b0f7ee45cc0d;concepts#16925b0f7ee45cc0d>\
<84,36,84,144;0,0,0,0;continous,arrow;f><;>\
<http://www.google.com/,text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee45d5d1;\
http://www.w3.org/2000/01/rdf-schema#subClassOf;3,0;\
concepts#16925b0f7ee45d5d1;concepts#16925b0f7ee45d5d1>\
<84,36,84,72,108,108;0,0,0,0;continous,arrow;f><;>\
<http://www.google.com/,text/html>
<http://www.conzilla.org/rdf/concepts#16925b0f7ee45dea5;\
http://www.w3.org/2000/01/rdf-schema#subClassOf;2,0;\
concepts#16925b0f7ee45dea5;concepts#16925b0f7ee45dea5>\
<84,36,84,72,114,78;0,0,0,0;continous,arrow;f><;>\
<http://www.google.com/,text/html>
```

TRITA-CSC-E 2006:040
ISRN-KTH/CSC/E–06/040–SE
ISSN-1653-5715

www.csc.kth.se