Dynamic Construction of context-maps for Netscape’s Open Directory Project

JOSÉ GIOVANNI PINEDA CASTELLANOS

Information Technology
Computing Science Department
Uppsala University
Box 311
751 05 Uppsala
Sweden

This work has been elaborated at the
KMR Group of CID/KTH in collaboration with Uppsala Learning Lab.

Abstract

The aim of this thesis is to explore the potential of visualizing information contained in a directory system, such as the Open Directory Project, by dynamically constructing context-maps on the server-side. These maps are constructed and subsequently presented by a Visualization Service through a Web Server. The purpose is to facilitate organization, exploration, extraction and presentation of information. The ideas and concepts implemented in this work are inspired in the spirit of the Semantic Web and a special tool - a concept browser - called Conzilla.

Keywords: RDF, XML, RDF/XML, concept browser, context-maps, Conzilla, Jena, DMOZ, ODP, Semantic Web, databases, Tomcat, Java servlets, server-side programming

Supervisors: MATTHIAS PALMÉR, AMBJÖRN NÄEVE KMR
Examiner: TORE RISCH, UDBL

Passed:
# Contents

1 Acknowledgements .......................................................... 4

2 Abbreviations ........................................................................ 5

3 Introduction ........................................................................... 6

4 Some important topics. ............................................................... 7
   4.1 RDF - a building block of the Semantic Web ............................. 7
   4.2 Conzilla: a concept browser .................................................. 11
   4.3 Open Directory Project: a human edited directory over the Internet 12
   4.4 Jena - Semantic Web Toolkit .................................................. 14

5 The State of the Art .................................................................. 14
   5.1 Visualizing RDF graphs ......................................................... 14
   5.2 RDF: Infrastructure for building the Semantic Web ................. 15

6 Problem Statement or Why do we need a Visualization Service? .... 16

7 Developing the Visualization Service ......................................... 17
   7.1 Design Issues and Considerations .......................................... 17
      7.1.1 Software architecture .................................................... 17
      7.1.2 RDF store requirements ................................................ 19
      7.1.3 Consideration for choosing a DBMS backend: Efficiency and Performance ............................................ 19
      7.1.4 Considerations for choosing a graphics format: PNG or SVG? .............................................................. 20
   7.2 Importing the data .................................................................. 21
      7.2.1 RDF Syntax issues .......................................................... 22
      7.2.2 MySQL™ specific features .............................................. 22
   7.3 Extracting the data .................................................................. 23
   7.4 Presenting the data .................................................................. 23
      7.4.1 Generating the Context-map .............................................. 23
      7.4.2 Outputting the result ....................................................... 25
   7.5 Development Environment .................................................... 26

8 Conclusions ............................................................................ 27
   8.1 Conclusions ........................................................................ 27
   8.2 Future Research .................................................................... 28

9 Bibliography ............................................................................ 29

10 Appendices ............................................................................ 29
   10.1 Appendix 1 ........................................................................ 29

References ................................................................................. 30
List of Figures

1 A simple RDF Statement ........................................ 9
2 A simple RDF Graph ............................................ 10
3 An arbitrarily long and complex RDF graph .................. 11
4 Example graph of the Structure model ....................... 13
5 Example depicting a graph from the Content model ........ 14
6 Schematic view of the DMOZ Visualization Service ......... 18
7 RDF Database schema .......................................... 22
8 UML Class Diagram auto-generated by the reverse-engineering tool EclipseUML[7] .............................. 24
9 Visualized information showing the related view, the directory view and the content i.e. the links ...................... 25
10 This figure shows exactly the same information as in figure 9 in a textual form ............................... 26

List of Tables

1 Storage manager comparison. All figures in ms .............. 20
1 Acknowledgements

A la familia Pineda Castellanos: Gratitud Eterna.

A Patricia Toledo por su cariño, comprensión y apoyo.

To Matthias Palmér, Ámirjörn Næve and Tore Risch for their guidance and critical contributions.

To Mia Lindegren from Uppsala Learning Lab for her encouragement.

To Mikael Nilsson for his indefatigable help in learning me the difficult art of mastering Debian and to Henrik Eriksson for his valuable advices.
2 Abbreviations

API     Application Program Interface
ARP     Another RDF Parser
CID     Center for User Oriented IT Design
DBMS    DataBase Management System
DMOZ    Directory Mozilla
FOAF    Friend of a friend
GUI     Graphic User Interface
HTML    Hyper Text Markup Language
HTTP    Hypertext Transfer Protocol
JDBC    Java Database Connectivity
KMR     Knowledge Management Research Group
NADA    Numerisk Analys och Datalogi
ODP     Open Directory Project
PNG     Portable Network Graphics
RDBMS   Relational Database Management System
RDF     Resource Description Framework
RDFS    Resource Description Framework Schema
RDF/XML XML Syntax Specification for RDF
RDQL    RDF Query Language included in Jena
RQL     The RDF Query Language
SQL     Structured Query Language
SVG     Scalable Vector graphics
UDBL    Uppsala Database Laboratory
UML     Unified Modeling Language
URI     Uniform Resource Identifier
W3C     World Wide Web Consortium
WWW     World Wide Web
XML     eXtensible Markup Language
XSD     XML Schema Datatypes
ZUI     Zoomable User Interface
3 Introduction

Modern society is characterized by easy and fast access to information. Due to the enormous increment of the amount of available information, we are facing what can be described as a tidal wave of information. In this paper we are particularly interested in the World Wide Web (subsequently referred to as the Web) as a source of information. Since its emergence at the beginning of the last decade, the number of WWW servers has increased from just a few hundred to about 35.4 million in January 2003[19]. From the perspective of the automated extraction of information, one of the main difficulties with the current Web architecture is that the information contained in it is largely structured purely for human consumption. Graphics and layout facilitate the processing of information by human brains but are completely useless for machines. The reason for this is that computers have no reliable way to process the inherent semantics in HTML documents.

As the detractors of the early Web correctly argued, the Web could never be structured as a well-organized library because of the absence of a centralized database and a tree structure. This means that you could never be sure of finding everything that is “out there”. Despite of the remarkably complete indices of information produced by search engines, there is still a clear necessity of structuring information in a more efficient way. Many users feel frustrated because they fail to find the precise information they are requiring. One approach to dealing with this problem is to categorize and describe the information itself, i.e. by creating data about data, which is commonly referred to as meta-data. There are several initiatives in this direction, the most commonly known is perhaps the Yahoo directory[36], which with limited resources, tries to manually categorize the entire Web. Obviously they will manage to describe a very small part of the Web and they will always lag behind. Another initiative, the Netscape's Open Directory Project[20] (ODP) uses a similar approach but relies on volunteers. Clearly these initiatives would benefit if their information could be combined and searched for simultaneously. This gives rise to a need for a standardized language for expressing meta-data.

The World Wide Web Consortium (W3C)[34] and the original creator of the Web, Tim Berners-Lee, strongly advocate what nowadays is called the Semantic Web[43] as a solution to these kinds of problems. It now appears likely that there will emerge a "second-generation Web" in which meta-data is going to play a key part. The Semantic Web will enable machines to comprehend semantic documents and data. Two important technologies for the development of the Semantic Web are already part of the current Web architecture: The eXtensible Markup Language (XML)[8] and the Resource Description Framework[25, 37](RDF). We strongly believe that this Web of well-structured, machine-understandable information will have a revolutionary impact on society by providing a substantial improvement in technologies such as e.g. e-commerce, e-administration, e-learning, e-science, organizational modeling etc.

At the present time, RDF and the Semantic Web provide us with a language which is machine understandable. Fortunately the ODP uses RDF for their representation of meta-data, but this does not necessarily imply usability and human understandable semantics though. In fact, the ODP user interface
is a list-based textual directory view where subdirectories can be navigated. However, the data itself is structured in the form of a graph, i.e. as a collection of nodes and arcs, and would, we argue, benefit from a more visual presentation.

In order to explore the potential of visualizing information contained in graph-like structures, we have developed a dynamic server-side visualization service for the ODP\(^1\). By visualization we mean the visual representation of data by means of computer graphics in a graph-like format (i.e. in the form of nodes and arcs), as opposed to the present textual presentation (we will subsequently refer to this service as the Visualization Service). For inspiration of what sort of visual presentations that are suitable we have looked extensively at the idea of context-maps.

Context-maps are a mix of mind-maps, UML-diagrams, concept-maps, topic-maps and other related graph-like presentations of knowledge\(^{49}\). A special tool - a concept-browser called Conzilla\(^{40, 50}\) - that supports navigation through an atlas of context-maps has been developed by the Knowledge Management Research Group (http://kmr.nada.kth.se/) over the last 4 years. Context-maps make a strict distinction between context and content. The purpose for this distinction is to enhance the overview of the data.

Conzilla is relevant for this thesis as a source of ideas and inspiration. In fact, all data presented by our Visualization Service is depicted in the form of context-maps. It is important to demonstrate that the subjacent ideas and concepts in Conzilla are also feasible in presenting the ODP and other similar structures.

Hence it is important to notice that Conzilla already is compatible with the Semantic Web and RDF. Context-maps are loaded and saved as RDF expressions and can be thought of as presentational information added on top of any other information expressed in RDF. Currently all Conzilla-maps are created manually, and since construction of maps is a time-consuming activity, just a few maps have been made. The Visualization Service is also aimed to put a remedy to this situation.

4 Some important topics.

In order to clarify our subsequent discussion this section enumerates a summary description of some relevant topics to this thesis.

4.1 RDF - a building block of the Semantic Web

The RDF is a language that enables the encoding, exchange, extension and reuse of structured meta-data on the Web. RDF is particularly intended for

\(^1\)In principle other information sources could be investigated as well, i.e. other initiatives than the ODP representing graph-like information could be presented along the lines described in this thesis as well, e.g. WordNet\(^{33}\), RPMind\(^{26}\), Musicbrainz\(^{36}\), etc.
representing meta-data about Web resources e.g. the title, authoring and copyrights of a Web page, users profiles, customers preferences, etc. RDF can also be used to represent meta-data about resources that can be identified on the Web even though these resources can not be electronically retrieved from the Web, e.g. items for on-line shopping, books in a library, etc. In other words, RDF is able to represent meta-data about abstract concepts outside the Web as well as physically existing objects.

RDF provides basic structural constraints that allow users to unambiguously express semantics defined by themselves. In addition, RDF provides a means for publishing human-readable and machine-understandable user-defined vocabularies. Structural constraints on meta-data provide for the interchangeability of separate packages of meta-data developed by heterogeneous communities of users, e.g. the Dublin Core Metadata Initiative[6], vCard[32, 31], Friend-of-a-Friend (FOAF)[1] etc. Since RDF is a common framework it is possible to make meta-data available to applications other than those for which it was originally created. This feature makes of RDF a powerful and versatile tool. The Resource Description Framework Schema (RDFS)[44] is a basic tool that enables users to define vocabulary, structure and constraints for expressing meta-data about Web resources.

RDF provides a simple and yet efficient way to make assertions about resources. In order to illustrate the way RDF works consider the following natural language statement:

John Smith created a Web page.

RDF is based on the idea that the resources we want to describe have properties which in turn have values and that resources can be described by making simple assertions about them. In RDF these assertions are called statements. In this case we want to state something about a resource: a Web page. In the RDF terminology the resource we are describing is called the subject. This particular subject has a property, which in the RDF terminology is called the predicate: the creator in our example. Finally this predicate or property has a value: the two words literal 'John Smith', which is called an object in RDF. The previous statement could be restated into the following English language statement:

http://www.example.org/index.html has a creator whose value is John Smith.

From this statement we can infer the following:

- The subject (or resource) is: http://www.example.org/index.html.
- The predicate (or property) is: creator.
- The object (or value) is: 'John Smith'.

In order to unambiguously describe resources on the Web, RDF needs a method to uniquely identify the subject, the predicate and the object inside a statement. In addition, RDF needs a mechanism to make statements about resources
machine-understandable, i.e. RDF needs a format for representing and exchanging RDF statements. These two requirements are satisfied by technologies that fortunately already are part of the actual Web architecture, namely the Web’s Uniform Resource Identifier (URI)[30] and the eXtensible Markup Language (XML)[8]. Notice that in our example statement we have already used the page’s URL. A URL is just a particular kind of URI. In fact, in RDF, even predicates and objects can be represented as URIs.

RDF represents statements as nodes and arcs in a graph. RDF graphs can be described as "labeled directed graphs" and represent only binary relations. In this notation, a statement is represented in the following way: a node (ellipse) for representing the subject, the node is labeled by a URI; an arc (arrow) for representing the predicate, the arc is labeled by a URI and is directed from the subject to the object; a node (ellipse) or a literal (box) for representing the object, the node or literal is labeled by a URI or by the literal value itself. Following these simple rules we can represent our example statement as shown in figure 1.

In our previous example we only considered an individual statement. Several such statements together form what is called an RDF graph where every statement is depicted as an arc. For instance, suppose we want to include additional information about our Web page, e.g. creation date, the language in which the document was written, copyrights, etc. This can readily be done by adding the corresponding objects, see Figure 2. This graph states that there is a resource, a web page called index.html, and it has a creator: John Smith (a resource itself in this case), it was created in August 16, 1999 and it was written in English.

Objects in RDF may be either resources, i.e. subjects, or constant values i.e. literals. However, literals may not be the subjects or predicates in RDF statements. This graphical notation may not be appropriate when handling large collections of statements since we could rapidly end up with immense and complex graphs. In such cases another kind of notation called N-triples notation can be used instead. See example2:

2One obvious disadvantage with this notation is that triples become illegible for humans as the URLs increase in length.

Figure 1: A simple RDF Statement.
The above triple\(^3\) represents exactly the same information as in figure 1. Regardless of the notation we choose to depict or represent a collection of statements the fundamental matter in RDF is the graph model\(^{[46]}\) of the statements. The notation itself is secondary. The very same information can also be written using yet another notation: the basic RDF/XML\(^{[23]}\) serialization syntax. RDF/XML encodes a RDF graph as XML elements, attributes, element contents, and attribute values. The following example listing depicts also exactly the same information as in figure 1 and the triple above:

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://www.example.org/index.html">
        <dc:creator>John Smith</dc:creator>
    </rdf:Description>
</rdf:RDF>
```

Considering that objects in some RDF statements may be subjects in other RDF statements, we are able to create arbitrarily long and complex graphs of statements. See figure 3. One such a graph is the Open Directory Project which consist of circa 16 million statements.

RDF statements are similar to other formats for representing information, e.g. entries in a simple record or catalog listing describing the resource in a data processing system, rows in a relational database system, simple assertions in formal logic, etc. Because of this property, information recorded in any of these formats can be treated as RDF statements, which allows RDF to be used as a unifying model for integrating data from different sources. This property enables RDF to serve as a potentially powerful bridge between different applications and platforms.

---

\(^3\)Another term for a statement. Subsequently we are going to use the terms statement and triple interchangeably.
4.2 Conzilla: a concept browser

A concept browser is a relatively new type of Knowledge Management tool whose purpose is to facilitate the organization, exploration, extraction and presentation of information that is structured in the form of a Knowledge Manifold\(^4\), by creating a survey of the information, i.e. by creating a birds-eye view of the information. The design principles for such a tool should primarily emphasize a strict distinction between context and content. The reason for this separation is that it simplifies the reuse of content across different contexts and for the sake of clarity. In order to clarify our discussion we first need to define a brief glossary\[40]:

- **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 (or component)**: information linked to a concept or a concept-relation.
- **Contextual neighborhood (of a concept or a concept-relation)**: context containing the concept or concept-relation.

Presenting information about things requires of some form of containing structure or context for the information that is to be presented. In one extreme there are paper-based information systems, such as e.g. a traditional dictionary, in which concepts are frozen into a single context. A dictionary uses an alphabetic

\(^4\)A Knowledge Manifold is a learner-centric educational architecture that supports customizable forms of inquiry-based learning\[41].
context which has the obvious advantage of making its components easily accessible by just following its alphabetical order. However, it has the drawback of making it difficult to extract additional related information because there exist no interconnections between components. A dictionary exhibits an example of what is called[40] a *totally disconnected (or discrete) contextual topology* i.e. that every single component is in absence of a contextual neighborhood. On the opposite extreme there are hyper-linked information systems, such as the Web, in which some concept may appear in many different contexts, whose number and form are constantly varying. This makes it hard to preserve the separation of context and content and results in the so called surfing-sickness because Web pages tend to be self-contained. The Web exhibits an example of what is called[40] a *dynamic contextual topology*.

The first prototype of a concept browser, called Conzilla has been developed since 1999 by the KMR group at CID at the Royal Institute of Technology in Stockholm. Conzilla is written in Java with a clear object-oriented structure. Conzilla is presently repositioned as an open source project at http://www.conzilla.org/ and can be freely downloaded from that site. Conzilla can work directly with the Semantic Web, both as a presentational and as an authoring tool.

### 4.3 Open Directory Project: a human edited directory over the Internet

The Open Directory Project ODP, also known as DMOZ,\(^5\) is a comprehensive, human-edited directory of the Web\(^6\). The directory is compiled by a vast global community of volunteer editors, approximately 56000 at the present time, and covers over 3.8 million sites. Information about these sites is organized in more than 460,000 different categories, including categories in a number of foreign languages, as diverse as Punjabi, Telugu, Basque, etc. The ODP is hosted and administered by Netscape Communication Corporation\(^7\) and has been developed in the spirit of the Open Source movement.

ODP editors collect information about the Web according to certain criteria stated in the Open Directory License. This information is subsequently structured in RDF using the Dublin Core schema as well as a schema developed by the DMOZ. This information is then released once a week, as RDF file dumps which are freely available for download from the ODP site. The list of sites, portals and search engines using the ODP RDF dumps include AOL Search, Netscape Search, Google, Lycos, DirectHit, AltaVista, HotBot, etc. The Open Directory Project is relevant for this paper because it constitutes the data source for constructing dynamic context-maps and its subjacent directory structure constitutes the context in which navigation will occur.

The ODP dump used by our Visualization Service is provided in two files, one file contains a description of the directory structure, i.e. the meta-data, we call this the Structure model. Let us take a detailed look at one example. Figure 4 shows a graph depicting a resource (a category in this particular case) called *Science/Astronomy/History/People*, it states that the resource has six

\(^5\) An acronym for Directory Mozilla

\(^6\) We are subsequently going to use the terms ODP and DMOZ interchangeably.

\(^7\) Part of the AOL/Time Warner Corporation.
different properties: type, catid, Title, Description, narrow and related. The property type indicates that the resource is of type Topic. The property catid tells us that the resource has a category identity whose value is the literal 534654. The properties Title and Description, which are defined in the Dublin Core namespace\(^8\) (see 4.1) tell us exactly what their names suggest. The property narrow indicates that the category Science/Astronomy/History/People has a subcategory, e.g. Science/Astronomy/History/People/Galileo. Finally, the property related indicates that the category Science/Astronomy/History/People is somehow related to other categories, e.g. Science/Physics/Relativity/People. Topic, catid, narrow and related are defined in the DMOZ namespace and have specific semantics. type is defined in the RDF namespace and also has a specific semantics. The structure is in fact much more intricate but this brief description is sufficient for our purposes. A complete description can be seen at http://rdf.dmoz.org/rdf/tags.html.

The other file holds the content itself, i.e. the relevant data for end users: links within each category that points to existing home pages. We call this the Content model. Figure 5 shows a graph from the Content model. It states that the resource Science/Astronomy/History/People has three different properties: type, catid and link. We have already described type and catid. The property link tells us that the resource Science/Astronomy/History/People has some links, i.e. some URLs that point to HTML documents that contain information about the

\(^8\)A set of names in which all names are unique.
resource. E.g. http://www.geocities.com/ziksby/index.html. This URL is itself a resource with three properties: Title, Description and Type (with value ExternalPage). The two former have also been described previously. The value ExternalPage indicates that this resource is in fact a link pointing to some specific document existing in cyberspace.

4.4 Jena - Semantic Web Toolkit

Jena[14] is a Java-based, open source application program interface (API) developed at HP Labs[10] for creating, manipulating and querying RDF Models. Jena includes facilities for the persistence of RDF data on a variety of database management systems which include PostgreSQL[22], MySQLTM[17], Berkeley DataBase[4], Oracle[21], etc. Other important features of the Jena API include ARP10 - which is a standards compliant, Xerces-based[35] parser for the full RDF/XML syntax for RDF - and RDQL which is an implementation of an SQL-like query language for RDF.

5 The State of the Art

5.1 Visualizing RDF graphs

The most simple approach to visualize a RDF graph is just to show the graph. This approach works well for small and hence trivial graphs, but it can quickly become confusing or even impractical for large graphs. A second approach is to
show one node and its neighbours at a time, i.e. stepping successively through triples. One disadvantage with this approach is that it may be a very slow and monotonous process, and it may show to little information at some times, specially in cases when the graph is large and complex as in our particular case. Two different efforts to deal with the limitations in these approaches are concretized in the following Semantic Web tools:

- IsaViz[11] is a tool that provides a visual interface for browsing and authoring RDF models represented as graphs. IsaViz is capable of importing graphs in RDF/XML and N-Triples (see section 4.1) syntaxes. In addition it is capable of exporting graphs in PNG and SVG formats. See section 7.1.4 for a discussion on this issue. IsaViz includes Jena and Xerces and requires GraphViz/dot (see 7.5) -in similitude with our own Visualization Service- and the Java Virtual Machine to be installed. The idea behind IsaViz is that of a metaphor of infinite virtual spaces that are explored using cameras that can be moved and zoomed. IsaViz provides a Zoomable User Interface (ZUI) which allows rapid navigation of graphs. By choosing to see a Radar View, a Graph View or a Global View over a graph the user can readily zoom in and zoom out over the entire graph and bring into focus different branches of the graph. This is a particularly useful feature while browsing large and complex graphs.

- BrownSauce[52] is a RDF browser which also utilizes Jena for manipulating RDF graphs. BrownSauce presents RDF graphs in a textual form, but the underlying approach for displaying the data is worth mentioning. The central idea behind BrownSauce is that of granularity, the problem with the triples approach is that it is too fine-grained. BrownSauce tries to remedy this by coarse-graining the data, i.e. breaking the data down into usable chunks or clusters. BrownSauce creates compound nodes. The intention is to get a reasonable amount of data that makes sense together, and display that.

5.2 RDF: Infrastructure for building the Semantic Web

RDF is the basic building block in the construction of the Semantic Web, but that task would be extremely more difficult without the existence of some basic frameworks to build upon. Besides the Jena toolkit there are other important efforts whose aim is to assist the work of the software developers community. The following enumeration is just a brief description of some current research projects.\footnote{For an authoritative, up-to-date and complete coverage in the field of the Semantic Web research check the following: \url{http://www.w3.org/2001/sw/Europe/}.}

- Redland[24] is an RDF Application Framework written in C that was developed at the University of Bristol. Redland provides high-level interfaces to instances of RDF models. Models can be stored, queried and manipulated in Java, PHP, Perl, Python and other languages. Redland provides modules for parsing and serializing which enables end users to read and write RDF syntax. Redland also provides for the persistence of RDF data through Sleapycat/Berkeley DB. Redland implements each of the RDF concepts in its own class via an object-based API.
• KAON[15] The KArlsruhe ONtology Project consists of a number of modules. The KAON Server module provides a RDF repository as an Enterprise Java Bean (EJB) that interfaces to J2EE application servers, with the data persisted in a RDBMS via JDBC. In addition, the KAON Server provides a high-level ontological interface which includes modules for accessing triple store or RQL-based\footnote{RQL is a declarative query language for RDF.} repositories. It also allows pluggable inferencers to be added for higher level logics. The 2002-10-02 version works with any SQL2-compatible relational database.

• Sesame\cite{27} is an Open Source RDF Schema-based Repository and Querying facility. It provides a generic API called SAIL Storage And Inference Layer. It interfaces between the RDF-specific methods and the database API. This API has methods for storing, removing and querying RDF in/from a repository. SAIL allows building of implementations on top of any kind of storage; be it a database, a file, or a peer-to-peer network. Current implementations are based on relational databases and Java graph models (memory storage) using plain files for persistent storage.

6 Problem Statement or Why do we need a Visualization Service?

Directory systems such as the DMOZ are a common part of the Web that have been in use for a long time. However, the majority of such systems are textual and they exhibit a “partially” disconnected contextual topology, which resembles that of our previous dictionary example in section 4.2. One major design goal in our system is to present the very same textual information in a visual form. In addition, we want to facilitate the discovery of hidden relations that are lost or somehow imperceptible in textual presentations.

The problems we want to focus on in this thesis include the following:

Graph presentation Information that is structured in the form of graphs or trees is simple to present node by node using lists and hyperlinks but this is not a very user-friendly approach. Human feel of overview and context of information is hardly encouraged. The process of creating subjective mental images is not well supported. Directory systems such as the DMOZ are ubiquitous, and most of such systems are displayed textually. This textual presentation effectively impedes the detection of relevant information. The visualization of information will assist end users in order to discover and process “hidden” relations that otherwise are imperceptible. The source of these problems is the lack of a clear separation between context and content.

Imperceptible information A graph contains information beyond individual edges. The relations between different parts of a graph is hard to comprehend if your interface only allows you to inspect limited areas.

Surfing sickness When you navigate a graph without the history of your navigation available you quickly lose focus.
This thesis works from the belief that end-users will benefit from visualization in many ways. End-users may sometimes not be sure of what they exactly are searching, the eye-bird perspective over the information offered by context-maps will make searching more effective and less frustrating.

7 Developing the Visualization Service

In this chapter we will describe all the different steps we took in order to develop a working Visualization Service, starting from raw, outdated RDF data-dumps. This process included the following: cleaning the RDF-dumps from illegal characters, converting the dumps into an up-to-date version of RDF, importing the data into a DBMS, extracting suitable chunks of data for visualization in response to end-users requests and finally generating views with an appropriate layout, in such a way that these views become explorable through interlinking. But first, we will start with some design issues and other considerations we were confronted to.

7.1 Design Issues and Considerations

7.1.1 Software architecture

Our Visualization System is intrinsically a multi-tier system which integrates different technologies that are freely available among the open source software community. The Java programming language is the glue that joins together all these technologies into one single cohesive system. Figure 6 depicts a schematic view of the system. When the system is put into execution a servlet is loaded into the Tomcat Servlet Container, the servlet's task is to listen, accept and process incoming requests made by HTTP clients on port 8180. When a request arrives, the servlet tries to establish a connection with the DBMS. This is done by means of the Jena API and the JDBC drivers, which considerably facilitates this task. The main reason for choosing to use the Jena API is that it is compliant to the W3C RDF specification, in addition, it is Java-compatible, it is supported by HP Labs and the Jena research group is among the leaders in the field of the Semantic Web. Once the connection is established, the DBMS handles the query and returns the required data to the servlet. The next step is to generate a drawing that represents the data, i.e. the context-map. This task is assigned to the dot engine (see 7.5). Grappa[48] is an API to the dot engine that provides methods which simplifies the creation, manipulation and display of graphs, nodes and edges and their attributes. It is important to remark that the Grappa API does not provide the layout, this function is fulfilled by the dot engine. The dot engine outputs a PNG file and a client-side map file which are directly placed into the Apache Web Server output directory. Finally the servlet dynamically constructs a HTML document for outputting both the textual information as well as the PNG-drawing and sends it to the HTTP client. This is done directly by the Tomcat Servlet Container or the Apache Web Server depending on if we choose to have the Servlet Container work as a stand-alone service or if we choose to activate the mod_jk Apache module, which enables the Apache Web Server and the Tomcat Servlet Container to work together.
Figure 6: Schematic view of the DMOZ Visualization Service.
7.1.2 RDF store requirements

We are going to concentrate in how Semantic Web data store relates to using RDBMs, specifically those licensed as Open Source or Free Software. The immediate question that arise is how to map RDF data and its schema to and from a relational database. The first step to take in answering this question is to take a look at the RDF data persistence requirements[42] and the features of RDBMSs that should apply for those requirements.

RDF persistence requirements should include at least the following:

1. Text searching: This requirement applies mainly for RDF objects whose value is a literal string. The RDBMS should facilitate standard word indexing, stemming, truncation and searching using typical operators as e.g. conjunction, phrase searching etc.

2. Support for URIs: An efficient support for URIs is useful since these are used to identify all terms in RDF graphs and are found in all parts of statements.

3. Support for Datatypes: RDF supports a datatype mechanism compatible with XML Schema Datatypes (XSD), which includes common datatypes such as e.g. decimal. It is expected that XSD will be more extensively used in the future.

4. Database tuning: The RDMSs should provide a way to evolve or update the database schema, such as e.g. subsumption support, i.e. the RDMS should provide us with a mechanism to add or alter tables smoothly. The reason for this is that RDF vocabulary descriptions may change in the future.

5. Triple provenance: It is often useful to be able to keep track of the origin of triples. This provenance information is outside of the RDF graphs and may be expensive if added by hand. Provenance information is desirable in supporting Web of Trust style applications.

7.1.3 Consideration for choosing a DBMS backend: Efficiency and Performance

Table 1 shows the results of a comparison between different database management technologies carried out by the Jena development team[51]. The tests were carried out on a small fragment of the ODP ranging from 20k to 200k statements i.e. using databases in the range of 50 to 80 MB. The timed tests included:

- Load time measured at several points during data loading. (ms per statement).
- QueryTree - forward query - find all category nodes below a given node in the category graph.
- QueryReverse - reverse query - given a leaf Topic node retrace the Topic links back to the root.
- QueryEq - literal match - find all Topics with a given Title string
<table>
<thead>
<tr>
<th>Layout</th>
<th>load</th>
<th>queryTree</th>
<th>queryRev</th>
<th>queryInt</th>
<th>queryEq</th>
<th>querySW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic/InterBase</td>
<td>16.0</td>
<td>1.2</td>
<td>3.7</td>
<td>25.5</td>
<td>3.1</td>
<td>79.8</td>
</tr>
<tr>
<td>Generic/PostgreSQL</td>
<td>10.2</td>
<td>1.4</td>
<td>7.6</td>
<td>3.8</td>
<td>7.3</td>
<td>14.0</td>
</tr>
<tr>
<td>Generic/MySQL</td>
<td>5.3</td>
<td>0.97</td>
<td>4.6</td>
<td>0.96</td>
<td>4.2</td>
<td>1.1</td>
</tr>
<tr>
<td>Berkeley DB</td>
<td>3.5</td>
<td>0.13</td>
<td>0.64</td>
<td>3.1</td>
<td>0.37</td>
<td>12.1</td>
</tr>
</tbody>
</table>

Table 1: Storage manager comparison. All figures in ms.

- QuerySW - literal range match - find all Topics whose Title starts with a given string.
- QueryInt - literal int range - find all Topics whose catid falls in a given range.

All tests were carried out on workstation class machines in the 700MHz to 900MHz range. The figures obtained are representative of small scale applications, where persistence rather than high volumes is critical. More serious scaling tests would need to be run on at least an order of magnitude more data. However, where comparisons between the 20k/200k samples were made, no major differential scaling effects became evident, so even these small tests have some validity. Some important conclusions made by the Jena development team include the following:

- The Berkeley DB storage manager (in non-transaction mode) is nearly an order of magnitude faster than most SQL database options. Transaction support is the primary overhead.
- For indexing the main statement table the two indices commonly used, namely subject, predicate and object, are indeed the best trade-offs.
- Stored procedures can save around 25% in load times.

We chose to use MySQL™ for our Visualization Service for various reasons. The first was a subjective one. We feel more comfortable working with MySQL™ partly because the documentation is more accessible and legible. The second reason was because of our previous experience with RDBMSs. However, as we can see in table 1, MySQL™ is also a good choice in terms of performance. MySQL™ is particularly effective for text-searching and integer-querying.

7.1.4 Considerations for choosing a graphics format: PNG or SVG?

Several graphics formats were considered for our purposes. Scalable Vector Graphics (SVG) [38] is a promising emerging technology, SVG drawings can be dynamic and interactive. SVG includes a rich set of event handlers such as onmouseover and onclick, which can be assigned to any SVG graphical object. In addition, as suggested by its name, SVG drawings are scalable, which means that the hole drawing or a part of it can be zoomed without losing information. All these characteristics make the SVG an attractive technology that is suitable for our purposes, since sometimes end-users will be faced with very complex drawings. Unfortunately, for the time being, SVG requires that end-users download and install plug-ins in their local hosts, which can be quite discouraging.
for many novice users. Portable Network Graphics (PNG)[39] is a mature and
deploy technology recommended by the W3C. PNG is supported by virtually
every Web browser. We chose to use PNG because it is widely available. SVG
could be a very good choice in future releases of our Visualization Service.

7.2 Importing the data

The first task was to create two databases. One database was intended for the
Structure model and one database for the Content model. The Structure model
is a RDF description of the DMOZ directory and the Content model is a RDF
collection of links contained in the directory. Both databases use exactly the
same database schema, which is an illustrative example of the versatility of RDF.
The databases themselves were created manually in the DBMS and the database
schemas were easily created programmatically by invoking the following Jena
API method:

```java
ModelRDB.create(jenaConn, "MMGeneric", "Mysql");
```

"Mysql" indicates that we want Jena to create a database schema specifically for
the MySQL™ DBMS. "MMGeneric" (Multi Model) indicates that the database
will be capable of holding more than one model. jenaConn is the connection
interface to the underlying DBMS.

The second task we were concerned about was to create an application for
populating these databases. Importing the DMOZ files into the databases was
unfortunately a tedious and time-consuming task. This was mostly due to the
fact that the DMOZ-files are provided in an invalid RDF-format. Another
reason was that at the present time the Jena API implementation is limited in
that it does not support bulk update/insert operations on databases. On PC
scale installations, Jena is expected to have a RDF-statements load times in the
range 20-50 statements/second. The DMOZ dumps correspond to roughly 16
millions statements, which means a total load time of around 4x10^5 seconds
i.e. approximately 4 to 5 days. Unfortunately, we could not measure the exact
loading time for the entire DMOZ dumps because the importing script crashed
several times. The reason is that the dumps are not compliant to the W3C
RDF specification. These problems will disappear in the future versions of the
Visualization Service.

Another limitation with the Jena API is that it is not able to handle big
amounts of data in memory, after the buffers become saturated, the import
script simply quits. (The size of an RDF dump varies between 350 MB and
1.3 GB). The solution to this problem was to write a small utility that splits
the RDF dumps into a number of much smaller files, not exceeding more than
for instance 2 MB. We must carefully check that every single file created also
becomes a legal RDF document. Subsequently, the original import script was
changed to essentially become a really large loop that reads the smaller files, one
by one, parses their RDF content and inserts the RDF data into some DBMS.
One important feature of the Jena API is that it automatically creates a data-
based schema which substantially facilitates matters for us. Inserting the data
into the database became a straightforward task achieved by simply invoking a
read-method defined in the API. Figure 7 shows a diagram over the database
schema. It is a very simple schema with just 6 relations.
7.2.1 RDF Syntax issues

Another obstacle we faced was the fact that the RDF dumps provided by the DMOZ are not in compliance with the present RDF syntax specification draft but rather with an earlier working draft. In addition, the RDF dumps also contain illegal characters. Thus the first task was to eliminate all the illegal characters. This was readily achieved by the following command:

```
tr -d '\000-\010\013\014\016-\037' dirty.rdf clean.rdf
```

The second task was to adjust the RDF dumps as to make them comply with the RDF syntax specification. Fortunately, this is a known problem previously handled by other persons. We downloaded two Sed scripts written by Sergey Melnik located at http://www-diglib.stanford.edu/diglib/ginf/download/dmoz/, modified them in order to fit our special needs (See Appendix 1) and applied them to the RDF-dumps.

7.2.2 MySQL™ specific features

In order to have the MySQL™ DBMS handle indexes correctly, it is necessary to increase the value of the key_buffer_size GLOBAL VARIABLE to at least 32MB. The default value is usually 16MB. This can be done by editing the configuration file my.cnf located at the /etc/mysql directory. This file contains a section called [mysqld] which should include the name-value pair "set-variable = key_buffer=16M". Just increase the value 16MB to at least 32MB. E.g. if your system has 256M RAM memory you can set the value to key_buffer=64M.

After that, do the following changes on table RDF_NAMESPACES (exactly in this order):

```
DROP INDEX RDF_IDX_NAMESPACES ON RDF_NAMESPACES;
ALTER TABLE RDF_NAMESPACES MODIFY URI TINYBLOB NOT NULL ;
CREATE INDEX RDF_IDX_NAMESPACES ON RDF_NAMESPACES(URI(250));
```

The reason for making this change is that some URIs are too long to be stored into a field of type varchar(250). The importing script was unable to handle that
situation, it just threw an exception and quit. This problem originates
from the database schema automatically created by the Jena API and will probably
disappear in future releases of the Jena API.

7.3 Extracting the data

Figure 8 depicts an UML class diagram of our Visualization Service. The
package se.kth.nada.kmr.dcm.db contains a Java class named CreateDBConnection.
This class does exactly what its name suggests; it has three methods,
the method getDBConn() is responsible for establishing the connection with
the DBMS, it uses the class DBConnection which is part of the Jena package
com.hp.hpl.mesa.rdf.jena.rdb. DBConnection in its turn uses the JDBC driver,
which ultimately is responsible for setting up the actual database connection.
Finally, the methods getContnetModel() and getStructureModel() are invoked.
These methods simply return two in-memory models of their database
counters.

After the connection is established, data can be retrieved and graphs, i.e.
context-maps, can be created. The package se.kth.nada.kmr.dcm.util is responsible
for both of these tasks. Similarly, the classes included in this package do
what their names suggest. ContentRetrieval retrieves metadata from the con-
tent database, LinkRetrieval retrieves URLs, i.e. links that point to existing
Web sites, RelatedRetrieval retrieves related categories to the category that is
currently displayed, StructureRetrieval retrieves information about the DMOZ
directory structure. All the previously enumerated classes make very extensive
use of the Jena API, which enormously simplifies the programming task.

7.4 Presenting the data

What are the aspects or facets of the directory system that our Visualization
Service should bring into focus? The DMOZ is a very large and complex graph
indeed, it exhibits a great quantity of cycles and paths. The DMOZ is usually
presented in textual form in most searching portals. These textual presentations
provide a Topic or directory-view\textsuperscript{13}, a related-view i.e. the related categories
in respect to the actual category, the link-view i.e. the amount of URLs that
belong to the actual category, etc. However, these views hide information that
could be relevant for end-users. Our Visualization Service emulates the very
same views. In addition, it provides a deeper related-view by going one step
further, i.e. by displaying the related categories to the actual related categories.
Compare figure 9 with figure 10. In this way a wider and more effective view
is provided. The “hidden” information, i.e. cycles and paths, suddenly becomes
apparent. This process of deepening could be repeated indefinitely until the
graph collapses. Still, the views have one Topic (or category) at its focus,
shown shaded as in figure 9.

7.4.1 Generating the Context-map

The next step, after all the requested data is collected, is to generate the context-
map. This task is carried out by the GenerateGraph class (see figure 8). This

\textsuperscript{13}By view we mean the extent or range of vision.
class makes use of the Grappa API in order to define a Graph class object, which we just choose to call the `graph`. This `graph` object will represent the entire context-map. Subsequently we attach nodes that correspond to the categories, subcategories and related categories included in the retrieved data. We also must attach edges that connect nodes to each other. This must be done carefully in order to preserve and reflect the relations that exist between the different nodes, i.e. categories, subcategories and related categories. Finally the `graph` is passed over to the dot engine which is responsible for the layout. The dot engine executes and outputs two files. One bitmap graphics format file (PNG in our case), and one map file. The map file is a so-called client-side map, which contains a list of the objects contained in the graphics file, their shape and coordinates. Web browsers use this information to make the graph clickable and thus giving our systems end-users the ability to navigate over the entire directory system. These files are directly saved in the Web server’s output directory.

### 7.4.2 Outputting the result

The package `sek.th.nada.kmr.dcm.servlet` is responsible for outputting the data to the HTTP client. This package contains just one class: `GraphViewServlet` (see figure 8). As indicated by its name, this class is a servlet and its main task is to respond to clients requests. The servlet communicates with the outside world by means of the interfaces `javax.servlet.http.HttpServletRequest` and
Figure 10: This figure shows exactly the same information as in figure 9 in a textual form.

class jsServlet.httpServletResponse. The servlet defines a PrintWriter object that is used to output HTML code:

```java
PrintWriter out = response.getWriter();
...
...
Header.buildHeader(out);
...
...
Footer.buildFooter(out);
```

Finally, the HTML document is sent to the client, the connections to the database are closed and garbage collection takes place.

```java
modelContent.close();
modelStruct.close();
```

7.5 Development Environment

We have chosen as implementation environment the Linux/Debian[5] operating system and the programming language Java™ Platform, Standard Edition (J2SE®) build 1.4.1_01-b01[12]. The Java technology is platform-neutral which will automatically guarantee the portability of our system.

All programming and testing has been carried out on an Intel Pentium III machine, equipped with 250 MB RAM, a processor speed of 600 MHz and a bus speed of 133 MHz. Other technologies used in the development work include the following:
• DBMS: MySQL™ version 3.23.54-log

• Jena API version 1.6.0.

• Apache Tomcat 4.0.4 Servlet Engine[2].

• Apache Web Server[29].

• JDBC™ Data Access API driver: mysql-connector-java-2.0.14[18].

• JAVA™ Servlet Technology[13].

• dot: dot is part of the GraphViz[9] AT&T Labs-Research[3] open source collection of tools for manipulating graph structures and generating graph layouts. dot is a batch layout tool for directed graph layouts. The input is a description of the graph in the dot language and the output is a rendering of the graph in a choice of vector or bitmap graphics formats.

• Grappa: Grappa is a Java graph drawing package that simplifies the inclusion of graph display and manipulation capabilities within Java applications and applets. Grappa is also provided as open source software from AT&T Labs-Research.

8 Conclusions

8.1 Conclusions

We have developed a Visualization Service for RDF-based Semantic Web data. It has the following properties:

Graph presentation: A graph-based method is used for visualizing the information. Using such a representation allows us to separate context and contents effectively. The context, i.e. the graph (ODP’s directory structure in this case), was separated from the content, i.e. the links to external Web sites. End-users navigate through the graphs, the links do not interfere with nor perturb the process of navigation. The links are not “part” of the graph, something which now becomes more evident.

Imperceptible information: A graph may be a very complex structure that is hard to comprehend if your interface only allows you to inspect limited areas. Our Visualization Service discloses “hidden” (or somehow imperceptible) information, since the visualized graph provides end-users with a wider perspective of the information. This goal is achieved by providing end-user with a deeper related-view. This property is completely absent in textual-based presentations.

Surfing sickness: When you navigate a graph without the history of your navigation available you quickly loose focus. Our Visualization Service presents data in such a way that end-users are all the time provided with contextual information about the way they navigate, i.e. end-users are at any time provided with their navigation history.
8.2 Future Research

Prof. James A. Hendler, a leading researcher of the Semantic Web at University of Maryland predicts "that in the next few years virtually every company, university, government agency or ad hoc interest group will want their Web resources linked to (Semantic Web) content — because of the many powerful tools that will be available for using it." [47] Despite this optimistic vision, the Semantic Web technologies are still in an early stage, i.e. ground-breaking and experimental. At the present time the main effort has been laid on its basic infrastructure. We now have a first set of languages, tools, and applications. The next step will be to provide inter-operable components that offer intelligent services to human users. The success of the RDF and the Semantic Web concept will depend on:

1. The development of applications that prove the applicability of the concept.
2. The availability of application interfaces, which facilitate the development of such applications.
3. Databases and inference systems that exploit RDF to identify and locate the most relevant Web resources [45].

In addition, many issues such as security, compatibility and usability will play a crucial roll. Such applications range from those that create and manipulate meta-data in RDF format to GUIs for editing RDF models, APIs, RDF persistence tools, etc. More research work in these areas is needed.

Returning to our own Visualization Service, we believe that major efforts should be concentrated on the following topics:

1. A similar visualization should be tested using other sources of data. E.g. WordNet, RPMfind, Musicbrainz, etc. WordNet is particularly interesting. WordNet organizes English nouns, verbs, adjectives and adverbs into synonym sets, each set representing one underlying lexical concept. The synonym sets are subsequently linked together by different relations. We believe that this kind of projects would notably benefit from visualization.

2. One evident improvement in future releases of our system should include an alphabetic sorting of search results. Also a more compact and intelligible arrangement of nodes should be encouraged.

3. We could take advantage of some capabilities such as *Find text as you type* included in some Web browsers such as e.g. Mozilla and Firebird. This could contribute to a more ergonomic navigation.

4. A full-text searching engine should be included. This could give end-users the possibility to customize and refine searchings. Searching should be possible both on the directory structure as well as on the titles and descriptions of categories and on the links themselves.

5. The feasibility of other Views should be explored. E.g. the al lang (alternative language) view.
9 Bibliography


10 Appendices

10.1 Appendix 1

#Sed script for cleaning the dmoz dump-files.
#Originally created by Sergey Melnik at Stanford University.
#Modified by Giovanni Pineda at Upsala Universitet.
s/ about=/ r:about=/
s/r:id=r:about=/
s/rdf"/rdf"/
/s/\TR/\RDF/\1999\02\22-rdf-syntax-ns#/
/s/<RDF /<r:RDF /
/s/<r:resource="">'(Top')(.*')/1/&dmoz;3;
/s/(r:ID="")'(Top')/(.*)'/1/&dmoz;3;
/<editor/is/(r:ID="")'(Top')/(.*')/1/&dmoz;3;
/<editor/is/(r:resource="")'(Top')/(.*)'/1/&dmoz;3;
/<editor/is/(r:resource="")'(Top')/(.*)'/1/&dmoz;3;

#No longer required by the new u8-files.
/s/<tag catid="\{0-9\}"/>

2i
References


