TECHNISCHE UNIVERSITÄT WIEN
VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

The Gnowsis

Using Semantic Web Technologies to build a Semantic Desktop

Ausgeführt am Institut

Information Systems Institute
Distributed Systems Group
der Technischen Universität Wien

Unter der Anleitung von

O.Univ.Prof. Dipl.-Ing. Dr.techn. Mehdi Jazayeri
und
Dipl. Ing. Gerald Reif

durch

Leopold Sauermann
leo@gnowsis.com

Hardtgasse 27/11
A-1190 Wien

____________________ ____________________
Datum Unterschrift

Abstract

Personal computers are gaining a more and more central role in our daily life. They are used to view images or DVDs, to listen to music, to communicate with others via email or instant messaging, and to organise our everyday life via calendars and contacts. Most of these tasks are done by standalone applications. This makes is almost impossible to link pieces of information across application borders.

The *Semantic Web*, which is currently under development by others, will be an extension of the current web in which resources can be described with metadata and information is given well-defined meaning. By transferring Semantic Web technologies to a desktop computer, this work shall enable users to access resources in fast and convenient. Information is represented and accessed in a uniform way, independent from the software application that generated it. This functionality we will call the *Semantic Desktop*. The aim of this work is to build *the gnowsis server*, the first implementation of a Semantic Desktop.

Therefore, we integrated data from different standard applications, transformed into the general *Resource Description Framework* (RDF) language. Additional data can be stored in a native RDF database. This allows the user to define needed links between resources such as a photo and contact information of a person. Links were stored and then used to navigate from one resource to another, across application borders. Through the use of URI identifiers and RDF, it was possible to extend this system to include different available resources and relations, taken from several standard applications.

Using the gnowsis prototype, which is a result of this work, applications have access to all important information stored in a single computer. Users are able to classify and structure their information in any way they want by creating bidirectional links between resources. A prototype information management tool *GnoGno* based on a wiki / weblog was built to explore this possibility.

Kurzfassung

Personal Computer (PCs) gewinnen immer mehr eine zentrale Rolle in unserem täglichen Leben. Sie werden verwendet um Bilder oder Videofilme zu betrachten, um Musik zu hören, um mit anderen per Email zu kommunizieren und um unser tägliches Leben mittels eines Kalenders und eines Adressbuchs zu organisieren. All diese Tätigkeiten werden meistens in verschiedenen Programmen erledigt. Es ist fast unmöglich, über die Grenzen von Programmen hinweg Information zu vernetzen.

Das *Semantic Web* wird eine Erweiterung des bestehenden *World Wide Web* sein, in dem Ressourcen und Dokumente mit Metadaten beschrieben werden und der Information ein Sinn gegeben wird. In dieser Arbeit geht es um die Verwendung von *Semantic Web* Technologien auf einem PC. Dem Benutzer soll es möglich sein, seine Dokumente schnell und einfach zu verwalten. Information wird in einem einheitlichen Format dargestellt, unabhängig von der Software die die Information erstellt. Diesen Ansatz nennen wir den *Semantic Desktop*. Das Ziel dieser Arbeit ist es, den *Gnowsis Server* zu erschaffen, die erste Implementierung eines *Semantic Desktop*.

Um dies zu erreichen, integrierten wir diverse Standard-Applikationen indem wir die Daten extrahierten und in die generelle *Resource Description Framework* (RDF) Sprache umwandelten. Zusätzlich verwendeten wir eine zentrale RDF Datenbank. So wird es möglich, Verbindungen zwischen verschiedensten Ressourcen zu definieren, wie etwa einem Photo und dem zugehörigen Adressbucheintrag einer Person. Diese Verbindungen werden gespeichert und können verwendet werden, um von einer Ressource ausgehend zu den verbundenen Ressourcen zu navigieren, über die Grenzen von einzelnen Programmen hinweg. Da das System auf URI Identifikatoren und der RDF Sprache beruht, ist es möglich, es auf andere existierende Daten auszuweiten.

Programme können den *Gnowsis Prototyp* benutzen, um Zugriff auf bestehende Information eines Systems zu erhalten. Benutzer haben die Möglichkeit, ihre eigenen Daten auf beliebige Art zu verwalten, in eigenen Strukturen und Hierarchien. Dies wird durch bidirektionale Links unterstützt. Das Informationsverwaltungsprogramm *GnoGno* wurde als Prototyp realisiert, als Beispiel für die neuen Möglichkeiten. Es ist auf einem Wiki / Weblog System aufgebaut.

Acknowledgements

It is my pleasure to thank the people and institutions that made this thesis possible.

First I want give thanks to my fiancée Ingrid Brunner, who was with me in good and bad moments during the last years and while writing this thesis. I just love you.

Many thanks go to my parents who supported me in every project I did, be it chaotic or good. My father introduced me to computers and my mother helped me emotionally and with her expert English knowledge.

I also want to thank Peter Pallierer who showed me a way how to extract data from Microsoft Outlook.

Special thanks go to my supervisor Gerald Reif. He introduced me to the Semantic Web. With enduring patience he taught me how to do scientific work, how to formulate arguments and to communicate a message. The good paragraphs in this work are the result of his influence; the chaotic parts belong to me alone.

I want to thank Prof. Dr. Mehdi Jazayeri and the members of the Distributed System Group who supported me with literature and pointed me to good ideas.

During the writing of this thesis, I was holder of a “Studienabschluss Stipendium” scholarship. It was funded by the Austrian *Bundesministerium für Bildung, Wissenschaft und Kultur* in cooperation with the *European Social Fund Austria.* I want to thank these institutions for their financial support.

The following Open Source projects where used to create the prototype, I thank their respective authors for their work and dedication to the Open Source Movement, without it, this would not have been possible.

- Apache Software Foundation:
    - Ant
    - Log4J
    - Xerces
    - JSTL Standard Tag Library
- Jacob by Dan Adler
- Jena – Semantic Web Framework by Hewlett-Packard Laboratories Bristol
- Jetty Web Server
- Junit testing
- MP3File by Jens Vonderheide
- WinAlwaysOnTop by Oliver Pfeiffer
- snipsnap.org by Stephan J. Schmidt and Matthias L. Jugel.
- Eclipse
- and code from other projects, embedded in the already listed

Dedication

Dedicated to
the LORD
and His Son Jesus Christ
the Savior

Who changed my life in the last years
and has carved the path
that I am following

For it is the LORD
who gifted me with
all I have
and all I can

Thank you, this one is for you

Contents

1 Introduction

"The dream behind the Web is of a common information space in which we communicate by sharing information. Its universality is essential: the fact that a hypertext link can point to anything, be it personal, local or global, be it draft or highly polished. There was a second part of the dream, too, dependent on the Web being so generally used that it became a realistic mirror (or in fact the primary embodiment) of the ways in which we work and play and socialize. That was that once the state of our interactions was on line, we could then use computers to help us analyse it, make sense of what we are doing, where we individually fit in, and how we can better work together."

Tim Berners-Lee,
http://www.w3.org/People/Berners-Lee/ShortHistory

Since the beginning of Personal Computers (PCs) in the early 80s computers evolved to powerful multipurpose devices. PCs are now used to play audio files and movies or to show digital photos, they are used to communicate with other people, and they are used to manage personal information such as calendars, contacts or to-do-lists. Photos we previously kept in physical photo albums are now stored in a digital album on a computer. Through the use of CD's and the innovation of MP3, a music collection is no longer on a shelve but on a hard drive. All digital data created by a single person – be it document, photo or video – can be stored on his computer. Since the constantly decreasing costs of storage devices there is no need to delete data anymore – when a disk is full, a new one can be bought at the same price but with double size. It seems that we will be able to keep all data for a lifetime. [Gem2002]

We should no longer ask whether we have enough information, we should rather ask if we can manage the information we have. In common operating systems, files are organized in folders. The folder structure is created by the user, based on some categorization scheme. Folders are associated to the date, type or the topic of the files they contain. We find them labeled "Business", "Private", "Car", "October2002", "Photos" or "Excel-Files". Using a categorization like that is easy to learn and can be found on the majority of private computers. A problem occurs if a single file can be associated with two folders. Should the photo of my son be stored in "Photos" or "Private" ? We often have to search in different folders to find the information we are looking for. It is frustrating to search for a much needed file, knowing that it's *there*, and not being able to locate it. In addition, different pieces of information are stored by different applications. There may be an appointment in Microsoft Outlook and a photo in a folder that was taken during the appointment. Although they are related, they cannot be put together. The user has to remember the relation and where the items are stored.

Based on the analysis of existing Personal Information Management (PIM) solutions this thesis aims to introduce a possible new solution that overcomes the existing application boundaries. The idea is to enable users to link objects with bidirectional hyperlinks. Categorization can then be done by linking related objects. Files that are about the same topic can be linked to each other. Therefore, one item can belong to multiple categories. The approach is to take existing technology and ideas from the WWW and use them for personal computing. The W3Cs Semantic Web Initiative proposed a foundation for processing metadata, the Resource Description Framework (RDF). Utilising this standard PIM resources can be linked across application borders. Therefore, RDF was used as basic technique when designing our prototype application.

Within this thesis a proof of concept prototype application was developed. The prototype is named *gnowsis*. It integrates files in the file system, Web resources and Microsoft Outlook contacts, appointments and emails. The user interface is embedded in the Outlook interface and in the Windows File Explorer. Items such as Outlook data and files can now be managed in a single category hierarchy. The user is able to browse through his information space across application borders.

The prototype is written in Java and based on the Jena framework. It is released under an open source license and can be used in custom applications. As the used technology is still fresh, it can also serve as inspiration for other projects.

1.1 Structure of This Thesis

The remainder of this thesis is structured as follows.

Chapter 2 gives an introduction to Personal Information Management (PIM). It defines the important terms, namely *resource* and *personal information*. It describes current scientific research and popular applications, the historic heritage we build on and a brief outlook on the future. The next step in development is to create a Semantic Desktop system.

Chapter 3 introduces the gnowsis Idea. It describes the basics of the gnowsis system, our approach to personal information management. First it gives an introduction to the goal of gnowsis and then it contains the core principles that were used. The system is based on the RDF data format. It is designed for a single user and contains a central database for the links. A discussion of all important parts of the system follows, with a focus on the scientific questions behind. The chapter contains the ideas that were needed before the system could be built.

In Chapter 4 a detailed description of the realized architecture is given. It contains the system architecture of the prototype. The subsystems are the Adapter Framework to integrate data, the gnowsis server to publish it and the GnoGno user interface to use it. Each subsystem is described with the implemented features and the used algorithms. The information management tool gnoBrowser can be found at the end of this chapter.

Chapter 5 discusses the experiences from using the system and lists the basic features. Before benefits are visible, a start barrier has to be crossed, a tutorial is given how to get accommodated. Then a possible usage is described, where to enter and how to manage information.

Chapter 6 concludes the thesis and summarized the contributions of this work. Possible use cases are given, as the gnowsis is a basis for future systems. Open issues are listed and the plans of the author to continue the project.

2 Personal Information Management

This chapter is an introduction to the field of Personal Information Management (PIM). At first a definition of terms is given. Then the different technologies needed to build the system are described. Hypertext and the Semantic Web are the basis of gnowsis. A short history of hypertext systems and the World Wide Web follows. Ideas from Enterprise Application Integration (EAI) were used for the architecture. The Science Fiction ideas of the cyberspace inspired future requirements. The features of current PIM tools are treated.

2.1 Personal Information

This thesis is about *personal information stored on a single personal computer*, especially the most common resources of daily computer usage: contacts, music, photos, appointments and documents. The term *personal information* is used for the data that a single person stores on his or her personal computer system. We do not use the term to describe information about a person itself. How this information can be stored and retrieved in a convenient way is known as *Personal Information Management*, or PIM.

Information management tools like “The brain” (2.4.5, below) are based on the principle of mind maps. These maps contain nodes that are connected through edges. The term “mind map” is no coincidence, as our human mind works in a similar way. Our long term memory stores entities of information and associations related to them. Recalling entities seems to be evoked by those activated in the short term memory. Associative learning is done by repeated activation of different entities in our mind. Children are taught the names of objects by pointing at the object and telling the name, the visual and auditory input are perceived together and associated. The data is used when a signal is sensed again. The perception system searches for similarities between sensed information and stored memories [Haikonen2003]. Thus we tend to categorize new entities into a system that we already have in our mind. Humans are excellent in classifying objects.

This human ability is a reason for software that allows the categorization of information. Information is classified according to a concept defined by the user. This could be to separate “business” from “private” or to sort files by their creation date. Information can then be found by browsing through the classes to find objects.

In a hierarchical file system, the folders represent classification schemes and files are stored in folders to express the classification. The same approach is taken in common email software to categorize email. Due to the lack of symbolic links, it is impossible to put a file in two folders in an MS-Windows file system. Using MS-Outlook, it is possible to associate a resource with more than one category.

If folders are used, the system does not know about the concept involved. For the computer, the folders are labeled with the literals “business” and “private”. But it is not able to understand the concept that is in the mind of the user.

Classification systems are used in many applications and they are the common technique used in PIM systems today. Metadata such as “which folder is this file in”, “creation date”, “author” is added to the objects. The task of classifying the objects and the decision where to store them is usually done by the user. The object is examined for certain properties, a decision is made in which class the object belongs and it is then stored in a folder that is associated to this class. For personal use, people are free to define their own classification systems and they use this freedom to develop creative structures that suit their needs. This works well because our brain is trained in classifying objects.

Systems have been built that allow an automatic classification; they decide themselves, what class a document belongs to (f.e. [Koch2000]). The common approach is to search for similarities and group similar resources together. We will not discuss the automatic approaches, as they are not popular in private use.

A simple example will be used in this thesis: A student, let us assume his name is *Leo*, attends the lecture "XML, XSL and Web applications" at the Vienna University of Technology. On his personal computer there is a file folder named "XMLLecture", the student uses the folder to store the course material, some XML source examples and solved exercises. There is a homepage about the lecture, where the student can find dates, news and downloads. The lecturer, *Clemens Kerer*, communicates via email with the students. The student, Leo, adds Clemens Kerer to his Microsoft Outlook when he received the emails. On Leo's computer exists a set of items that are related to the same idea, attending a university lecture.

But with today's common software architecture, the classification stops at the application border. The file-system does not know about the classification used in the email program and vice versa. The user has to rebuild the same structures in different applications. These structures are related: they are derived from the concepts in the mind of the user.

After this introduction to PIM approaches, we will focus on the data items that are to be managed. In the next chapter the concept of resources will be described.

2.1.1 What is a resource?

There are different approaches to define what a resource is. Herbert Simon has suggested a synthetic unit (the CHUNK), Keith Devlin has defined a mathematical approach using the term "infon" [Devlin1991, Simon1974]. In this thesis, the term *resource* is used for digital resources. Here is a list of examples:

- Files
  - Documents
  - Scientific papers
  - Photos
  - Digital books
- Emails
- Diary entries
- Contact items in an Address Book
- Appointments
- Reminders
- Notes
- Bookmarks in a Web browser
- Web pages

The question arises where to separate a resource, as it may include other resources; similar to a document consisting of chapters. In addition to the theoretical definitions we need a set of practical rules. We created the following rules to identify resources and to work with them.

A resource has to be *distinct*. There is a clear distinction of one resource from another. The distinction can be found in the structure of existing applications, how the application separates its data in files or database entries. It is also possible to identify resources in the user interface. These are items in lists or viewed in separate windows.

A resource is *identifiable*. Based on the fact that a resource is distinct from another, there is a set of properties used to distinguish the resources.

Additionally to these rules, resources have some general properties.

A resource *represents a thing or a concept*. Resources have some meaning, they can be used as a symbol to represent some physical object or an idea. Furthermore, they may contain information about the represented entity. A contact item in an address book serves as a symbol for the living person and also contains the telephone number.

Resources contain data in a *structured* and *unstructured* way. In a structured representation the different properties of a resource are stored separately from each other, like the `firstname` and `lastname` field in a address book, whereas unstructured data is usually represented as a long text. Structured data can be easily extracted from software systems and therefore computations in this work are only done on structured data, often with the intent to find some unstructured data.

A resource can also be described by *metadata*; structured information about the resource itself. The author, title and last modification date are common metadata of a document.

The term resource is used as a unifying concept for all the different items that are stored in a personal computer, all data can be seen as a collection of resources. From our “XML lecture” example, the following items are resources:

- a folder to store files
- a PDF tutorial file
- the website at the university
- an address book entry for the instructor, “Clemens Kerer”
- The schedule of the lecture with date, time and place in the Outlook calendar.

2.2 History

The field of information management has a long history and interesting examples can be found in the pre-computer area, for example carbon-paper and the standardization of paper formats. Focus is given to hypertext systems, as the gnowsis is based on hypertext and inspired by the people that developed it.

In 1945, Vannevar Bush wrote the famous article “As we may think”, where he described the visionary system that he called “Memex” [Bush1945]. The definition that he gave was important for many systems to follow:

“Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and, to coin one at random, "memex"

will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory."

Vannevar Bush, [Bush1945].

In this short paragraph we have a compressed description what a Memex should do. Bush based his ideas solely on analog devices, running on punch cards and using microfilm as storage, it was never actually built. Today we notice how his vision becomes reality, the personal computer is very close to what Bush had in mind. Not all books and records are stored in a PC, but we are close to it. This is no coincidence, as this article has influenced the development of the computer industry. Still, there is work left to create the "intimate supplement to memory". Bush requested that all documents have numbers, identifiers, in this thesis we will use URLs to fulfill this part of the Memex vision.

In 1960, Ted Nelson created the idea of the "Xanadu" system and he never stopped developing it. Xanadu was a predecessor of hyperlink systems, Nelson coined the term "Hypertext". He wanted to actually build the Memex, he wrote the article "As We Will Think" [Nelson1972], an answer to Vannevar Bush's "As We May Think" [Bush1945]. At first Xanadu was a mockup prototype to show the idea, today there is a windows prototype available for free download[1]. It never ignited the revolution that was intended by Nelson [Nelson1960].

In 1992 the World Wide Web launched, created by Tim Berners-Lee. The Web grew at a very fast rate. It has changed society; information is used in a different way than in the pre-web era. Before the web lifted off, Berners-Lee programmed the "Enquire-Within-Upon-Everything" system. "Enquire" (the short version of its name) was a tool to enter information about people, projects, hardware resources and how they relate to each other. It was used at CERN to store administrative data about equipment and software. It was created out of a certain need:

"What I was looking for fell under the general category of documentation systems – software that allows documents to be stored and later retrieved. This was a dubious arena, however. I had seen numerous developers arrive at CERN to tout systems that 'helped' people organize information. They'd say, 'To use this system all you have to do is divide all your documents into four categories' or 'You just have to save your data as a Word Wonderful document' or whatever. I saw one protagonist after the next shot down in flames by indignant researchers because the developers were forcing them to reorganize their work to fit the system."

Tim Berners-Lee
[Timbl2000, p. 17]

These were the requirements that lead to the WWW. Berners-Lee wanted to have a version of Enquire that could be used in a distributed scenario, where people could access his database through a network. Other people should be able to publish their

[1] http://xanadu.com/cosmicbook/CosReadInstaller.exe

data on their own servers. The existing documentation systems did not match the requirements. Hence the need to create something new, a distributed version of Enquire that we know today as the World Wide Web. [TimblFAQ]

The interesting fact is, that the Web had its revolution in the distributed world but the topic of personal information management remained the same. The field of "documentation systems" is still a vivid arena with many competing companies. Enquire had the ability to handle bidirectional links and these links were labeled. The features of it were lost in the process of creating the Web – but the need for them remained. In the Web, documents are just "linked" with the anchor tag of HTML, but it is not clear how the objects relate to each other.

The problem of metadata and labeled links was identified and is now tackled by the Semantic Web Initiative. The other problem – bringing the features of Enquire back to life – is the topic of this thesis.

2.3 The Semantic Web

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

Tim Berners-Lee, James Hendler, Ora Lassila, "The Semantic Web", Scientific American, May 2001

The Semantic Web is a vision of an improved kind of Web with enhanced functionality which will permit semantic-based representation and processing of Web information [SemWeb2001]. The World Wide Web Consortium (W3C) has proposed a series of technologies that can be applied to achieve this vision. The Semantic Web extends the current Web by giving the web content a well-defined meaning. XML is aimed at communicating and storing data so that different systems can understand and interpret the information. XML is focused on the syntax of a document and it provides essentially a mechanism to declare and use simple data structures. However there is no way for a program to actually understand the knowledge contained in the XML documents.

Resource Description Framework [RDF] provides a common framework for expressing the semantic information so it can be exchanged between applications without loss of meaning.

RDF is based on the idea of identifying things using Web identifiers (URIs), and describing resources in terms of simple properties and property values. A basic **RDF statement** consist of three parts. First the **subject** – a resource that is to be further described. The second part is the **predicate**, what property of the subject the statement further describes. The third part is the **object**, a value of this property. The object could either be a literal (a number, a name) or another resource, identified by an URI. Subject and predicate are always an URI. The three together, subject, predicate and object, form a statement that is called a **triple**. This enables RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values.

To describe the information "This thesis is written by Leo Sauermann", the sentence is divided into subject, predicate and object. The subject is the thesis, the predicate is the authorship and the object is Leo Sauermann. All the three can be identified by URIs, here is an example:

```
Subject:   http://www.gnowsis.com/wiki/GnowSis/LeoSThesis
Predicate: http://purl.org/dc/elements/1.1/creator
Object:    http://www.gnowsis.com/wiki/Main/LeoS
```

These three URIs are globally unique and valid identifiers for the respective concepts behind, the predicate was taken from the ontology standard Dublin Core, this will be explained in detail in chapter 3.6.

RDF can use the XML vocabulary to express these triples in a way that can be exchanged between systems and used for automated processing. There are other notations to write RDF, for example N3 [Timbl1998].

The W3C provides a recommendation to define the vocabulary which can be used within RDF documents. This vocabulary language is called RDF Schema [RDFS]. Using it, the structure of RDF documents can be described. RDF Schema can be used to define properties and classes of the web resources. Similar to XML Schema which imposes specific constraints on the structure of an XML document, RDF Schema provides information about the interpretation of the RDF statements. Based on RDF Schema, simple ontology systems can be described.

Any other ontology language may be used. The Web Ontology Language – OWL – builds on top of RDF(S) to provide a language with both well-defined semantics and a set of language constructs including classes, subclasses and typed properties. OWL can further express restriction on membership in classes and restrictions on domains and ranges of properties. [OWL]

In order to get acquainted with the basics of RDF and RDFS we recommend the following articles to the interested reader:

- RDF Primer [RDFPrimer]
- RDF/XML Syntax Specification [RDFXML]
- RDF Vocabulary Description Language 1.0: RDF Schema [RDFS]

2.4 Commercial Tools

In 1980, vendors were offering to sell “documentation systems” to the CERN, and these were not fitting the requirements (see Section 2.2 above). Today systems are different and the cited problems have been identified and solutions should be available. Many useful applications are competing on the PIM market, created by commercial companies and research groups.

The market of commercial tools for Information Management offers a broad range of products. To fulfil the needs of companies, Lotus Notes and Microsoft Exchange are in competition. For a single user, smaller companies are offering solutions. The common products and their features will be described in the following pages.

2.4.1 Common Features and approaches today

Information management tools enable users to store documents, find documents, link documents, classify them. Some may also be used for idea management. The goal is to support the user in entering information and finding it. There are different common features in this field, products usually support one or more of these.

Indexing of text files is a common feature, together with the ability to identify important *keywords*. Resources can be *linked* and *categorized*. Other products support the management of *ideas, little notes* or *to-do* lists.

The software systems follow two basic approaches. The *monopoly* approach is to import all data from other systems and restrict the user to the new software system. Programs that supply an "all in one" database containing "all knowledge" fall into this category. The other basic approach is *integration*, where data stays in existing applications and the existing applications are enhanced with the new features. A mixture of both approaches is also possible, as many integration tools build their own index files.

The following list of information management tools is a small excerpt to show what products are available at the time of writing. All of these products may be bought by individuals at reasonable prices or are without license fees. They should be suited for the task of our example student, to organize his lecture and connected resources. For companies and larger organizations, there are more sophisticated tools, more expensive and often client-server based. We will not include these here.

2.4.2 X1

http://www.x1.com

This is a very popular program that creates an index on all resources a typical person uses. It integrates with all common applications and provides an easy user interface. The system was formerly known as "find".

It enables the user to search for resources and open them, also short abstracts are created. There is an integration to web search engines. The product has been rated as very useful by popular media like the "PC Magazine", "The Wall Street Journal" or "C|NET".

2.4.3 Coredge Logik

http://www.coredge.com

"Logik" is able to index common file formats and MS-Outlook-like systems. The key benefit of Logik is that it processes all resources and automatically generates short abstracts of the resources, it also able to identify the topics of a resource. The user can also classify resources manually. It integrates with common applications.

Through this automatic extraction of topics and keywords, it is more sophisticated than other indexing systems.

2.4.4 Mind Model

http://www.mindmodel.com

Mindmodel is in the tradition of small database systems like AskSam, dBase and MS-Access. It has a sophisticated user interface and the entered data can be interlinked. The approach of mind model is monopolistic.

2.4.5 The Brain

http://www.thebrain.com

This software creates visual, web-like structures, similar to mind mapping. The user interface of "the brain" is simple and elegant, the main way of interactions is by drag-and-drop. It uses its own storage for notes and relations and can integrate files and web-pages.

The user interface design and webpage design of the company are creative.

2.5 Other tools

A look at the market shows us that there are many more competitors. The products tackle the problem of finding information and most of them are based on a few simple principles. Mind maps, text indexing and notes are well known approaches. So we move on to more specialized tools that can be used to manage information. Following are those who have influenced the development of gnowsis.

2.5.1 Topic bound tools

There is a multitude of tools that can handle information on a specific topic. Software to organize "your music collection", "your stamp collection", "all of your photos", "your star wars collectible figurines collection" etc.

These tools are very useful in their respective topic but do not integrate into other systems like MS-Outlook.

A nice example of a topic bound tool is the PhotoFinder System by the University of Maryland. It is a great tool to organize photos, people shown on photos can be linked to the photos and there are many other useful features. [Kang2000].

2.5.2 Weblogs

A *weblog*, or *blog*, is a diary that is published on a website. The action of entering new content is called "blogging". A "blogger" is a person who keeps such a weblog. On February 18th 2002, *wired* wrote that blogging crossed the tipping point from a "self contained community" to a major movement [Blog2002].

Blogging is hype in 2003, there is a multitude of weblogs and there are more to come. The main feature of a weblog is that the blogger is able to publish a message by entering a text and pressing a button. This simplicity is striking. The weblogs have two major purposes: publishing ideas and keeping an archive. Weblogs contain many links to previous weblog entries by the same author or related weblogs by other authors. Often a weblog entry discusses a website, then there is a link to the website. Searching in a weblog can be done by date or keyword.

An interesting fact is that bloggers often use their own weblog to search for information they have used some time ago. As the weblog is always online, a blogger can use it as an online information management tool.

Weblogs can be integrated into the Semantic Web, as Steve Cayzer has outlined in a paper. His focus was on publishing existing metadata [Cayzer2003].

2.5.3 Wikis

A *wiki* is a web content management software that allows users to enter information in a simpler syntax than html. The key feature of a wiki is that it creates hyperlinks to other pages in the wiki system automatically. There are different approaches, the most common is that words with two capital letters, like DiplomaThesis, are automatically hyperlinked to the page that has the name DiplomaThesis.

These keywords are called "Wiki Words" for their use in a wiki or "Bumpy Words" or "Camel Case" – because of the two bumps. Web pages in a wiki are written as plain text files through a web interface, the text can contain special formatting characters and Wiki Words. Every page has a unique name, a Wiki Word identifying it. If a Wiki Word is entered in a page and no corresponding page with the name exists, a new page can be created automatically by clicking on the word or a special tag near to it.

Most public wikis are open for everyone, every user can change all content. Wiki systems are extensively used for collaborative documentation of large systems. Wikis are also a good tool for information management, because they offer bidirectional links and basic idea management. Dozens of different implementations exist in all major programming languages, some applications have modified the original wiki guidelines [Wiki].

2.5.4 Blokis

Finally, the merger of a wiki and a weblog is called a *bloki*. The idea management part of the gnowsis is based on a bloki, we enhanced it to enable links to resources outside the bloki, to any file or object (see 4.6 “gnoBrowser”).

2.6 Scientific Fields

There are several approaches to the field of personal information management. As the above commercial examples show, text indexing, mind maps and text notes are features that customers pay for.

Here is a list of the most influential papers, other references are at the according places throughout this work and can be found in the literature appendix.

2.6.1 User Interaction

In the area of user interaction and user interface design, there are a few projects that are similar to the gnowsis project.

Haystack, by the MIT Computer Science and Artificial Intelligence Laboratory, is a tool designed to let an individual manage his information in a way that makes the most sense to him. It has a monopolistic approach and is a replacement for many applications including word-processors, email clients, image manipulation, instant messaging and other functionality. The system contains many revolutionary ideas. It is a research prototype. Everything in the system is a resource and operations can be done on the resources. It contains a classification system for resources. Many user interaction features are implemented as context menus, that appear when clicking the right mouse button. [Haystack]

Another interesting prototype is the *MyLifeBits* by Microsoft Research. It is a lifetime store of multimedia data, based on the assessment that all information a single person reads and hears can soon be stored on a portable device. Every day a person consumes audio, video, text and other media. If a hypothetical disk of one terabyte per year is available, it would be possible to store all this multimedia on it. The MyLifeBits paper describes a concept how to manage this huge amount of media, how to classify and retrieve the data. [Gem2002]

Forget-Me-Not by Lanning and Flynn of the Rank Xerox Research Center is an approach to note information on small mobile devices. Resources are represented by small icons and have a history. Events in time are represented by sentences constructed from these icons. For example “Mike went to the kitchen” is represented by an icon of Mike, the "@" representing the location property "being at" and a coffecup icon for the kitchen. This was an inspiration for the wiki-text parser and also for using symbolic resources for real things or concepts. [ForgetMeNot].

In "Finding and Reminding Reconsidered" [Fertig1996] Fertig, Freeman and Gelernter discuss a preceding paper. The common desktop metaphor of storing information in named files and organizing these in hierarchical folders is just one way of handling data. They insist that research about the way users acquire, organize, maintain and retrieve information has to include most current approaches. Especially they discuss a paper by Barreau and Nardi [Barreau1995], who evaluated folders and files. The new metaphors could be *"The Semantic File System"* by MIT, *"Dynamic Queries"* by Ben Shneiderman or their own system, *Lifestreams*. Lifestreams, by Freeman and Gelernter is based on the idea that files are identified by their creation or modification dates and other categorization methods, usual file names and folders are not needed anymore. [Free96]

2.6.2 EAI and Middleware

Middleware is, for short, "a technology that brings together systems that were never meant to work together".

In the Enterprise Application Integration (EAI) field, the term "Middleware" describes systems that handle "Data Integration", with the goal to access data from a heterogeneous set of systems and publish them in a uniform way. Data Integration systems often have an architecture like a wheel, with a central hub and spokes, that connect the hub with the different systems. [Hol2003]

The gnowsis system was designed to be the middleware on a single user's desktop.

2.7 Future

The Semantic Web is not established yet. Today we only have the seeds planted and we see projects growing, but the main act is yet to come. With the Semantic Web as described above, it is theoretically possible to represent all structured data and metadata available in the World Wide Web in a uniform syntax and with agreed semantic meaning. In the Semantic Web data are not just represented but also made available for all connected clients and servers. If all data are presented in a uniform information space, new ways of data visualisation and browsing will evolve. This will have an impact on how we work with information, how we communicate and how we interact socially. Like the telephone and the WWW, it is not possible today to know what the changes will be, but the Semantic Web will be a step in a direction that leads us to a global, ubiquitous information space.

In science fiction literature, the *cyberspace* is the name for such a global network, the term created by William Gibson in his 1984 Novel "Neuromancer".

"Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation, by children being taught mathematical concepts. . . A graphic representation of data abstracted from the banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the non space of the mind, clusters and constellations of data. Like city lights, receding. . ."

William Gibson, "Neuromancer" [Gibson1984]

Computer science is a fruitful field for Science-Fiction authors. They have created stories around a global network of information, the cyberspace. It all started with the short story "True Names and Other Dangers" by Vernor Vinge in 1981 [Vinge1981]. Then came the famous "Neuromancer" by William Gibson, who coined the term "Cyberspace" [Gibson1984]. Other authors like Bruce Sterling and Neal Stephenson joined in and the separate branch of "Cyberspace Literature" evolved.

The cyberspace idea is nearly the same in all stories. It is a global network of information, a future extrapolation of the internet. There is a three dimensional visualization of this information world, this visualization is like a parallel world, real objects have their counterpart in cyberspace. A company has a real world office and a cyberspace office, a three dimensional webshop. It is created and shaped by humans and is the place, where all important information work happens. The user interface to the cyberspace is a portable hardware system that creates a three-dimensional illusion, it is a networked device like a web browser today. The main user interaction paradigm is that the user "travels" to the desired information and "is" at the place where the information is stored and "sees" the information. To find other information, the user may follow a kind of link or has to travel, using a virtual transport system.

We are able to take this as an inspiration for our own work. The elegance and clean architecture of a global three-dimensional cyberspace could be a goal for the Semantic Web movement, a system where all information of the world is represented and can be manipulated in a three dimensional space.

The gnowsis system was inspired by cyberspace and is designed to be one basis for future research in this area. When all data is already in a unified format and can be identified with URLs, it remains a rather simple step to add the 3d-position metadata to existing resources. Then we could create a visual browser to our personal information.

To do this, at first all data has to be accessible in a uniform syntax and protocol, especially on the personal desktop.

2.8 Missing Link

How will the current Semantic Web projects lead to the planned global network? Today, the popular projects in the community are focused on enterprise, server or collaborative systems, the goals are set on a high level and the intended customers are large companies.

On-To-Knowledge[2] was a major Semantic Web research project in Europe. The application focus was knowledge management in large and distributed organisations. It has been the biggest Semantic Web project in Europe so far. The few commercial Semantic Web companies also focus on organisations. Ontoprise[3] GmbH positioned itself as a provider of semantic technologies and solutions for large companies.

An alternative direction was chosen by us:

If the goal is to have a global Semantic Web,
one building block is a Semantic Desktop,
a Web for a single user.

Nearly all information we see on web-pages and in electronic documents today has been created by people using personal computers. The PC is the place where most personal data is stored and the major interface to the web. Information stored on a server is usually manipulated through interfaces that are executed on a PC.

To create all this content, we have to bring the Semantic Web to the personal computer, to the person in front of the screen. The use of ontologies, classifications and global identifiers in normal desktop applications has not happened yet.

Let us look at the university example again, how can the instructor of a course publish the day, time and location of his lecture? If this data is already entered in his calendar application, it would be great to export the existing information to the university website. A semantic search engine could gather information about courses and create a sample timetable for students in the fourth' semester. Students may import the time and date directly from the website to their corresponding date book application. A student could use a course finder tool to search for available lectures in this semester, especially courses that are part of the student's curriculum and wait to be done. Today we copy and paste the text values from one application to another, this will change sooner or later.

If the idea of the Semantic Desktop should turn into a real system, we have to start building it today. The gnowsis is a research prototype in this direction. Existing Web technologies can be transferred to personal computers and used for Personal Information Management. The next chapter describes the gnowsis Semantic Desktop.

[2] http://www.ontoknowledge.org/
[3] http://www.ontoprise.com/

3 The Gnowsis Idea

This chapter introduces the ideas behind the gnowsis system. The philosophy is to take existing Semantic Web technology and transfer it to a personal computer. Resources on the computer are addressed with a URI scheme and their metadata is converted to RDF. Existing applications are included in this architecture and not replaced. A close look is taken on the meaning of an URI, in the context of using it to identify ideas and concepts.

3.1 A Semantic Desktop

The Semantic Web is a way of enriching the existing WWW with data. This technology may also be used on a personal computer. Similar to the Web, a server application publishes the available data. This local Semantic Web Server can be used to access the information stored on the PC. Files and other resources can be used like Web resources. They are identified by their URI and links can be created from one document to another. The existing data is available in the RDF syntax and can be used by any application. Interaction between the applications is simplified by these standards.

By taking the Semantic Web technologies and using them on a personal computer, a semantic layer in system architecture is created. Parallel to the existing access to resources using already installed applications, they can be accessed through the Semantic Web server. The resources are identified independent of their file format. Based on this layer, a system-wide classification system can be created. Concepts like “private” or “business” are represented once in the system and can be reused in all applications. The class “my XML Lecture” is represented similar to a group in a web directory (like yahoo), the class can contain files, emails, webpages, etc. All resources can be identified by their URI and can be opened using a browser like program. A crawler similar to google can be built on top of such a system to create a full text index of all existing text data. The index

The links between local resources are bidirectional, this is an extension of the common WWW principle of unidirectional links. A relation between two resources can be entered at one place into the system and can be found from all connected resources.

As shown by these examples, bringing web technologies to a personal computer can be used in many ways. Accessing the data on the computer through the interface of a Semantic Web server creates a new perspective of existing data, they can be addressed by URIs and organized in web directories etc. To build such a system, a few prerequisites are required. In this chapter the technical basis for such a Personal Semantic Web is described.

3.2 Principles

3.2.1 A single user on a single system

The system is built to be a Semantic Web for a single user, containing the resources of this user. It should serve the needs of this single user. Some other PIM systems are designed for a single user, see [Gem2002] and [Haystack].

The restriction to a single user simplifies the system, as no multiuser support is needed. The application is executed on a single computer. It is not distributed and lacks user authentification and authorization mechanisms. There are no collaboration features embedded in the system, if data has to be shared with other users, it has to be exported and imported manually. The RDF data is stored in a single database. The contents of this database is declared as belonging to the single user.

Just as every person is an individual and can be identified, a gnowsis system is identified by its unique *DNS hostname*. Through this globally unique hostname, it is possible to create unique identifiers for all data on the machine. If the computer does not have a unique global hostname, a workaround hostname has to be built. This workaround hostname has to be in a domain that is owned by the user and then prefixed with an invented name. In the development of the system, the hostnames “test.gnowsis.com” and “leo.gnowsis.com” were used, hosts for these names were never registered and do not exist, but because the author owns “gnowsis.com”, no one else has the right to use them, so they are unique.

In a future ubiquitous internet, the unique hostname may be used to contact the host and query RDF information from it. The single user is identified by a system-wide URI, which leads us to the question of identifying resources.

3.2.2 Everything has a URI

It is vital to the system that all resources have distinctive identifiers. A uniform way to identify resources has already been conceived in Bush’s Memex article [Bush1945]. Today’s computing is impaired by the fact that there is no identification system for resources on a personal computer. For PIM, we need to reference objects across application borders. There are different existing systems to identify resources. For example, a file is identified by its local filename. An email is identified by its message-id in the header. Identification of a book can be done through its Library of Congress Control Number (LCCN), International Standard Book Number (ISBN) and International Standard Serial Number (ISSN). So there exist many methods of identifying resources and also different identification methods for the same thing.

If a reference to a resource has to be stored in a system, the type of resource it is usually known at design time. To build an online bookshop, a decision is made if ISBN or ISSN is stored in the database behind. The column in the database will be called “ISBN_nr” and the data type will be able to store the number combination. The system is limited to the use of ISBNs and cannot be changed.

Using a Uniform Resource Identifier (URI) to identify a resource has several advantages compared to other reference systems like plain ISBN. A URI is globally unique. It is not restricted to a local database but can be used worldwide. It contains information about the identification. Instead of using multiple identification system like filenames and message-ids, we decided to transform every identifier into a URI scheme. To store a reference to any resource, just the URI is needed. Information about resources can be queried based on the URI of the resource. There is no need to add context information about the type of resource and where to find it, everything can be extracted from the identifier itself.

The basic syntax of a URI is the following [RFC2396]:

```
<scheme>://<authority><path>?<query>
```

In this thesis the parts of a URI are also described with the following common names:

```
<protocol>://<hostname><path>?<query>#<anchor>
```

Resources stored on the user's computer are identified by URIs containing the user's *DNS hostname*. Therefore, the use of the hostname makes the identification of the resources globally unique.

Some resources on a desktop computer can already be identified in this way. If a ftp server is running on the host, a ftp URI can be used to identify files. On our example system, no ftp server was installed so a workaround was used. A file:// path was mapped to a path on a local hard-drive. The path "file://leo.gnowsis.com/data/" is mapped to the "My Documents" folder of Microsoft Windows. This is similar to a file share configured with the "My Documents" folder as local root and "data" as public share name. Files in the "My Documents" folder can be addressed with a global unique URI by this method.

Therefore existing URI schemes such as "file://" or "http://" may be used to identify documents on a web server. Many resources on a PC are not accessible through an internet protocol but also need to be identified using a URI. The emails stored in Outlook are an example, they are all stored in one single file, the "outlook.pst". We had to define a custom syntax for these resources. This syntax was inspired by other common URI notations. It has the scheme `"rdfp://"`, which does not exist[4]. To distinguish the different applications, a prefix has been used at the beginning of the path in the URI. The first part in the path identifies the application that is host of the resource. For Outlook, we used "`msoutlook`" as identifier (see Figure 1). This custom URI scheme of gnowsis is useful to identify resources. Metadata can be extracted and the resources can be identified for the execution of commands but there is no `"rdfp"` protocol to retrieve the resources themselves.

The URI scheme to identify Microsoft Outlook resources is based on the `rdpf` scheme. We are aware that this is a proprietary way to identify Outlook resources with URLs. Our scheme may change, if a better standard is available.

[4] The "imaginary" RDFP protocol may turn "real" very soon. Members of the W3C SWAD group are aware of the lack of protocol and when you read these lines, something like RDFP may already be defined in a standard.

Outlook URI scheme

```
rdfp://<hostname>/msoutlook/<type>/<identifier>
```

Description of the parts:

<hostname>	the hostname of the computer
<type>	the classname, f.e. “mail”, “contact”, “appointment”
<identifier>	the unique number identifying the resource, this number is extracted from MS-Outlook.

Example:
```
rdfp://leo.gnowsis.com/msoutlook/contact/
00000000ECD4B99358B9814B9DAFE2255CD8AE9AA4822000
```

Figure 1 Outlook URI Scheme

```
The single user:
rdfp://leo.gnowsis.com/~user/leo

The Lecture-Folder on the Harddisk:
file://leo.gnowsis.com/data/uni/xml/

A solved exercise:
file://leo.gnowsis.com/data/uni/xml/lab05/lab5.xml

The tutorial file:
file://leo.gnowsis.com/data/uni/xml/skript/xsl.pdf

The Outlook Contact of the lecturer, Clemens Kerer:
rdfp://leo.gnowsis.com/msoutlook/contact/
00000000ECD4B99358B9814B9DAFE2255CD8AE9AA4822000

The Outlook Appointment of the lecture date:
rdfp://leo.gnowsis.com/msoutlook/appointment/
00000000B2CDC30BFF2EED4ABA9C61436A07FE33C4002000

The homepage of the Lecture:
http://www.infosys.tuwien.ac.at/teaching/courses/XML/
```

Figure 2 XML Lecture example URIs

The single user of the system is also identified. Again the <hostname> is included here, the <username> is a unique name for the user, usually the first or last name.

```
rdfp://<hostname>/~user/<username>
```

Another special application is the gnowsis system itself and its need to store configuration and runtime information. An example is the database file location stored in the configuration which has to be identified. For these resources, a similar scheme has been used. The server uses unique <identifier> parts in this URI.

```
rdfp://<hostname>/~server/<identifier>
```

Examples are listed in Figure 2. Note that the files are identified on a path starting with “data”. This path is mapped to the “My Documents” folder of Microsoft Windows. The lecture folder “xml” is a subfolder of “uni”, which contains all university lectures and is a subfolder of “My Documents”. This structure is a result of how the student organises his material, other students have different structures.

3.2.3 Everything is RDF – transforming information

The gnowsis system fulfils the role of an EAI middleware by integrating metadata into a central hub. This is done by transforming the metadata of a resource into a common data format. All resources that are listed in chapter 2.1.1 should be available in this central information hub. Transforming all data into a single format allows us to access any data through the same interface. There is no need to implement data transformation routines in every program, the data can be accessed through the gnowsis server.

We chose RDF as data format for all information stored in the central information hub. As Paul Korzeniowski pointed out, RDF is suited for data integration in Enterprise Application Integration [Korz2003]. Since this work is based on Semantic Web technology, so RDF is the right syntax for metadata. The notation of RDF is not important. There are two major standard notations to write RDF. RDF/XML uses XML files to note RDF. “Notation 3” (N3) is a simpler plaintext format. Both formats are known to all major RDF tools and therefore the gnowsis system will handle both formats. Internally, the system can use custom representations, like a SQL database.

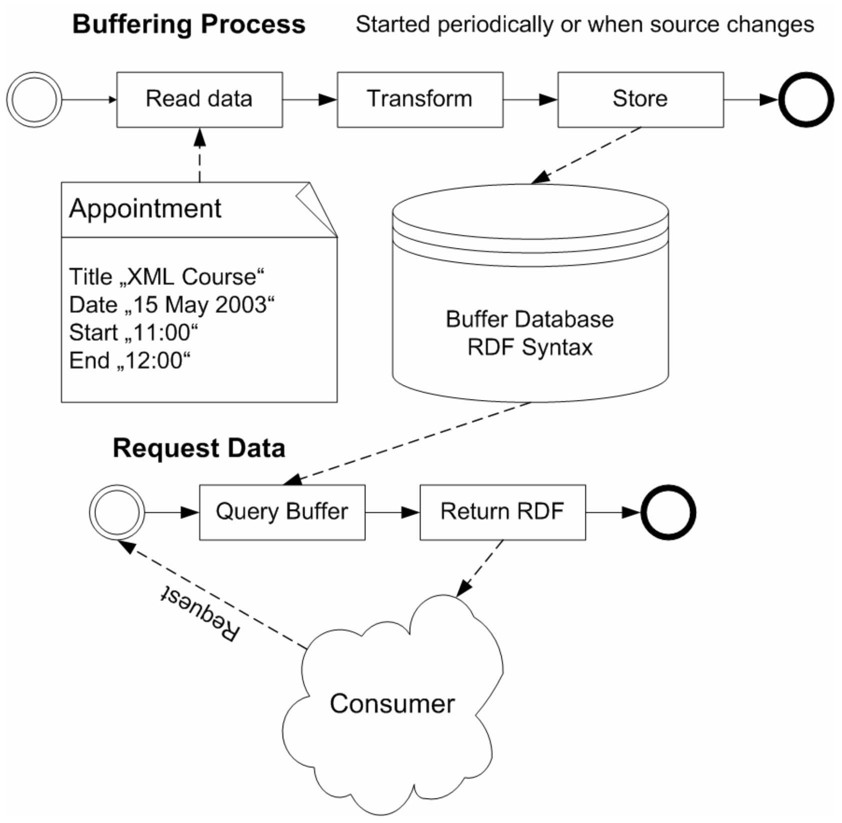


Figure 3 Transformation and Buffering

There are two approaches to the transformation of data. The first approach is shown in Figure 3 Transformation and Buffering. The task is divided into two processes, data extraction and data retrieval. All needed data is extracted from the source applications, transformed to RDF and buffered in a database. Updates are necessary when resources change, so resources have to be watched. Extracting and converting data into RDF is a simple process. Existing RDF databases may be used to store the converted data. The database has a native RDF structure, suited for this task. We evaluated different storage systems in [Sauermann2003] and decided to use the internal Jena database. Querying the information from the database is fast, all data is in a uniform database and can be indexed. Unfortunately all resources have to be continually watched for changes, which complicates the program. As a consequence all data is stored both in the database and in the source files. Although disk space is cheap, this doubles the needed storage space.

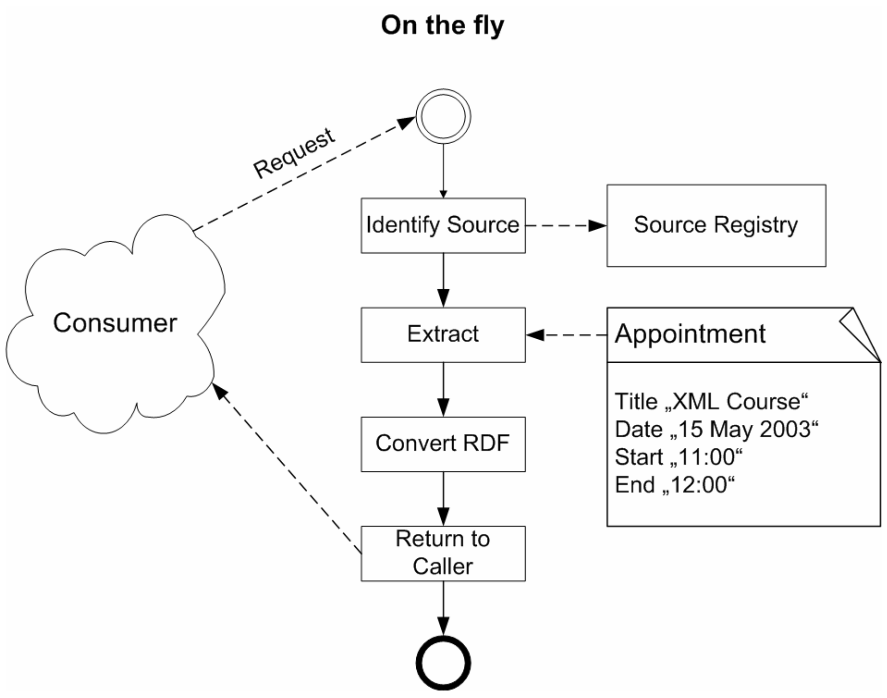


Figure 4 On the Fly Extraction

In the second approach the transformation to RDF is triggered on demand, when metadata about a resource are queried, see “Figure 4 On the Fly Extraction”. First the query is analysed to identify what resource is requested. Then a registry of all data sources and some algorithm is needed to deduce the correct data source of the resource. The resource is read and the queried data extracted. This data is expressed in RDF and returned to the caller. This approach slows down the response time of the system, as the deduction and transformation process takes place at each request. This method has the advantage that transformations are only made on demand. The original data is used during extraction, so the results are never outdated.

The first approach has been tried out by the author, because it appeared to be simpler for this prototype application. A test utility was written to convert metadata of a set of MP3 files to RDF and store this data in the central repository. But when the conversion utility was analysed, it proved that the second approach, doing the conversion on the fly, is also possible within the resources of this project, and therefore the second approach was chosen.

Converting all information of a single computer into RDF on the fly is a cleaner architecture for data integration than buffering. Data is not stored redundant and converted only when needed.

In the next section we will focus on the source of all data, the applications that manage resources on a personal computer.

3.2.4 Hosting applications

On a normal desktop system a set of applications is already installed. These applications can manipulate the resources stored on the system. For example, Microsoft Word can manipulate a Microsoft Word Document. These applications are usually the best way to manipulate their resources, they can extract metadata, create new resources, delete them, print them, view them, etc.

We decided to call them *hosting applications*. This term will be used throughout this thesis. They are responsible for their resources. Each resource has to have a hosting application, or it cannot be manipulated by the user. It is important that existing hosting applications are not replaced by the gnowsis, they should be enhanced. Existing data is reused and can remain in its native format and be manipulated through its hosting application. The gnowsis just adds a way to identify any resource and extract RDF data from it.

Gnowsis depends on these hosting applications, the APIs of these applications are used to extract metadata of resources and also to manipulate resources. If the user wants to open a resource for manipulating or viewing, the hosting application is called to do this.

If a hosting application is responsible for opening a resource and cannot be used to extract metadata, third party software is needed. This had to be done for MP3 files: The Winamp software by Nullsoft plays the MP3 files but has no API to extract metadata, so another tool to extract ID3 metadata from the files has been used [MP3File].

It is a central design principle in gnowsis not to replace hosting applications but to integrate existing applications into the Semantic Web. Replacing all applications would be a monopolistic approach.

3.2.5 Central storage

Bidirectional links are a central feature of the gnowsis and the only clean architecture for bidirectional links is a central storage system. In addition to the existing applications, the gnowsis is also a hosting application that manages some data. So a central storage system is required for the gnowsis.

The central storage, also called central *repository*, contains bidirectional links, configuration data of the gnowsis server, ontologies and other data that can not be retrieved from a resource. It can be used by other applications as a convenient RDF data storage server, if needed. The central repository is based on an RDF database system.

An alternative approach would be to store the metadata in the resources themselves. This approach was evaluated by the author in spring 2003 and has been discussed in October 2003 on the RDFIG mailing list[5]. The current state of the art for embedding RDF metadata is:

- Using Adobe XMP[6]
- Write RDF/XML data somewhere hidden in the file, use a plaintext-parser to extract it.
- Modern file systems have metadata extensions,
    - ReiserFS 4
    - HFS and HFS+ for MacOs have data and resource forks in files
    - NTFS has streams (nobody uses them)

[5] http://lists.w3.org/Archives/Public/www-rdf-interest/2003Oct/0188.html
[6] http://www.adobe.com/products/xmp/main.html

- Microsoft has a tradition of announcing this feature for new operating systems, “Longhorn” will have a database enhanced file system
- The METS standard [7]
- Access files through a WebDav server

Another possibility is using a local WebDav server. This possibility was evaluated intensively for gnowsis, as it allowed a very elegant way to embed the use of metadata into the system. An additional drive would show up in the file system, where folder’s would not match physical file folders but metadata classifications. The categories are represented as folders and files could be found in more than one folder, similar to MIT’s Semantic File System. WebDav and gnowsis would be tightly coupled. There are open source WebDav implementations[8] available that could be modified to fit into this scenario.

The alternative approach of embedding metadata in the resources was not used by the author as it implies disadvantages:

- The programming effort is much higher, as most applications do not support custom metadata in their data files by default. Also, if applications support custom data, each application has a different programming interface to do so, which results in a high programming effort.
- If an operating system supports the attachment of metadata to files, what about resources hidden in files ? Think of the many Outlook resources that are hidden in one single Outlook.pst file.
- There is no central index for bidirectional links
- The gnowsis and also the hosting application would have write access to the same file, possibly at the same time, which may result in concurrency conflicts.
- The user should use the hosting application to modify standard metadata where possible. The metadata is then extracted by adapters from the resource files.

3.2.6 Associative linking of resources

The human brain has a good support for associative memories. The concept of associative storage is a good archetype for PIM software. Mind Map tools are based on this idea. The resources in the gnowsis system can also be associated to each other using RDF triple statements. Resources from different applications can be linked to each other. The borders between applications are crossed by this way.

A resource can be linked to more than one other resource and be classified in more than one category. Connecting the resources can be done with the Dublin Core property “relation” or by any other property. Expressing these links in RDF is simple; a triple connecting subject to object. The problem is to provide a user interface to create the links and to give them a meaning. Ontology systems like RDFS are needed to define useful predicates linking an object to a subject.

[7] http://www.loc.gov/standards/mets/
[8] http://www.webdav.org/projects/

3.3 Ontologies

In the Semantic Web, ontologies are used to describe metadata about resources. Common ontology systems are thesauri and classifications, possible properties of resources can be described in an ontology. The two common ontology languages are RDFS and OWL. We focused on RDFS to support ontologies in the gnowsis system. There are four areas where ontologies are needed.

First, the server itself depends on custom data formats. Internal operation data (like counters, etc) and the configuration files were defined using ontologies. These were expressed in RDFS language and stored as RDF/XML files in the server system.

The second area is data extracted from hosting applications. Microsoft Outlook contains contacts, emails and appointments. An ontology describes what data can be found inside a resource of the hosting application. For each different hosting application, a separate description is needed. These ontologies were created based on the ideas of Becket and Grant in [Bek2003]. In fact, most of the problems that Becket and Grant identified for RDBMS were also evident in adapting hosting applications. Their chapter "5.5 Choice of property names" refers to the problem of expensive queries, this issue is addressed in the chapter 4.2.4 "Mapping and Ontology" below.

The third area of use are public ontologies. They describe common things like people, projects or documents and have been created by experts. Published in the internet, they are to be used by any interested party. Public ontologies are important for a worldwide standardization in the Semantic Web. In this stage of the Semantic Web, there are only a handful of them. We chose two ontologies to be integrated in the gnowsis:

- "Friend Of A Friend" by Dan Brickley and Libby Miller. [FOAF]
  It contains classes for people and common data like organisations and documents. The idea of FOAF is to describe individuals and their relations.

- "Dublin Core" [DCMI]
  Created by the Dublin Core Metadata Initiative, it is used to describe documents in detail. It is an exhaustive vocabulary that contains terms for predicates like author, subject, title and copyright of a document.

Finally, the fourth area are custom ontologies by the user. Each user may chose to create a new ontology that suits his special needs. Custom classes like "has to be done soon" or "related to work" could be examples. For testing purposes, a custom ontology for PIM was started, we called it "Vienna Core" (a pun with Dublin Core). Gnowsis can be used to integrate custom ontologies but there is no ontology editor included, so the authoring is still a problem of its own.

Ontologies can be either included from RDFS/XML text files or stored in the central repository. The gnowsis system is able to import ontologies with a web based configuration tool.

3.4 Collaboration

Creating an associative web of resources is like weaving the thought structure of the user into the computer, the resulting web is subjective and is best used by a single user. Because this data is not easy to communicate and to simplify the first version of the system, no collaboration features were included in the system. The system is isolated and there is no support to share the metadata with other gnowsis systems. Also, there is no automatic way to publish the metadata or to send it by email etc.

Before ontology metadata can be communicated to other individuals, some collaboration rules have to be established. Usually, the participants in the communication have to agree to a common ontology. Individual metadata then either has to be converted to this common ontology or the same common ontology has to be used by all participants.

If in the future such a common ontology exists, the gnowsis system can be extended with a collaboration module. Import and export functions are already provided, there is a RMI interface to the storage system and all information is globally unique through the URI, therefore collaboration is possible in principle.

3.5 URL revisited

On the Web, it is a well known fact that a URL identifies a resource. Evaluating the parts – protocol, host and path – it gives an indication of where to find and how to retrieve the resource. The URL specification is one of the three pillars that the World Wide Web is built on: URL, HTTP and HTML. Internet addresses have left the boundaries of computer screens. Mentioned in TV commercials, printed on business cards and a label on consumer products they are integrated in everyday life.

It seems that it is clear what a URL is and what it does, but this is not the case. Since the invention of the URL, its nature is subject to discussion. In this section the use of URLs in gnowsis is described.

3.5.1 URL and URI – Location and Identification

In his book “Weaving The Web”, Tim Berners-Lee described that at first, there was only the concept of the URI (at this time meaning “Universal Resource Identifier”, this changed to “Uniform Resource Identifier”) which was an identifier and locator for a document. Then after a discussion with the IETF, the term URL was introduced, the *Uniform Resource Locator*. The difference is, that an identifier does not change whereas a locator changes when the location of the document changes [Timbl2000, p. 67].

In gnowsis, the URIs are identifiers and locators. A URI is parsed and evaluated for two reasons. First, to know where the resource identified by the URI can be found and second, where metadata about this resource can be found. The URIs are parsed and if a corresponding hosting application for the resource can be found in the gnowsis system, metadata can be extracted using this hosting application.

This approach of dereferencing URIs to find out where the resources are (host) and how to access them (protocol) is one of the reasons why the World Wide Web is such a success. There are some problems with URIs regarding continuity but at the moment a URI is the best way to identify a resource.

The question remains, what exactly a URI is and what it points at, this question is known as the "URI crisis".

3.5.2 The Uri crisis

When enabling users to classify resources is the goal, first the classes have to be defined. In the introduction, we gave an example of how classification can be done. Files can be divided into the classes "private" or "business", expressing this fact by storing files in different folders labeled "private" and "business". In a Semantic Web system, these categorizations can also be implemented. A representation for the classes is needed in a RDF ontology. In RDF schema, we can define classes and relate objects as belonging to the class. A prerequisite is to identify the class with a URI. So we need URI identifiers for the classes "business" and "private" and we need to use them in the software system. But what exactly is a URI and what does it identify, this question has to be answered before it can be used.

In the end of the year 2002 and through 2003 we have seen the so called *uri crisis*, or *identity crisis*. There have been discussions about this problem in the Technical Architecture Group (TAG)[9] of the World Wide Web Consortium and there are some interesting articles about this topic [Pep2003] [Clark2003] [Timbl2002].

A single URI can have different meanings. It can have four aspects, as identified in [Booth2003]. Using the definition of the concept of "love" as an example, the different aspects are shown.

What does **http://x.org/love** identify?

- The *name* "http://x.org/love" itself. This is a string that conforms to the URI syntax.
- A particular *concept* of love.
- The *web location* from which a document can be obtained.
- The *document* instance that describes the concept of love, it can be retrieved from the location and then further processed.

When the URI is used in a Semantic Web application, it may be used in all of these contexts. The TAG group has been concerned about distinguishing the *concept* from the *web location*. In the public discussion, two approaches for a solution crystallized. We will describe them here in a short way, to illustrate the approaches.

Different URIs

The first approach is to use different URIs for the different contexts the resource is in. Basically, a new system is needed to identify concepts. URIs identifying web locations already exist. The concept-URIs may have different parts that distinguish them from web locations: a new protocol has been proposed (`rdfp://`, `info://`, etc). This would imply that also a new protocol has to be implemented, which is not practical. The fragment identifier (#) at the end of a URI could be used to identify the concept (…love#concept). Fragment identifiers are often used in

[9] Search for „HTTPRange-14" at http://lists.w3.org/Archives/Public/www-tag/
or search for „Uri crisis" at http://lists.w3.org/Archives/Public/www-rdf-interest/

systems to identify parts of documents and can thus be used to identify resources. So fragment identifiers are no stable solution. A special hostname of a server that just hosts concepts can be inserted at the beginning of the URI: “http://concept.x.org/love” in contrast to “http://x.org/love”. This would imply the installation of new servers.

All these approaches suffer the same disadvantage: the URLs used to identify the concept may already exist in the current web and no real distinction can be made between concept and document location.

Different Context

The second major approach is defining different contexts for a URI. A discussion about this solution can be found at [Pep2003]. Basically, the URI is either used as a “concept identifier” or “resource identifier”. Applications will handle a URI differently depending on the context it is used in, concept or resource. The problem with this approach is, that the current RDF standard cannot be used to implement it. Also, it is not obvious on which rules the different context should be used.

A solution for this problem has not been accepted by the community because of the philosophical implications involved. What exactly “is” a website, what “is” a concept and how do we distinguish both? Some companies have bound their company tightly to their web presence. So what is the difference between “www.amazon.com” and the Seattle based company, especially from the view of the user? The sentence “I buy at amazon” and “I buy at www.amazon.com” are not easy to distinguish.

"It needs must be that what can be spoken and thought is; for it is possible for it to be, and it is not possible for what is nothing to be. That is what I bid thee ponder."

Parmenides of Elea [Parm]

So what “is” amazon and what is that is spoken and thought? The term “www.amazon.com” is a symbol we use in language and it represents something. The meaning of symbols has been discussed for a long time, this question is of philosophical nature and will not be addressed in this thesis. Instead we have to find a practical solution for the usage of URIs to represent concepts and things.

A URI can be used to identify a thing.

To identify the book “Moby Dick” by Herman Melville, we can use the page on Amazon where the book is on display. Entering the URL into a browser opens a page where the book is shown. This example is included in an article of Tim Berners-Lee about the URI crisis [Timbl2002]. In the sentence “I want to read Moby Dick”, the Amazon URI may be used to identify the book. There are several URIs that point to a webpage of Amazon where the book is on sale, so it is ambiguous. For a human reader, the URI works – to learn what it identifies, the user can open it in a web-browser. It is also rather simple to identify persons, if they are stored in an address book application. Then the URI of the address book entry is taken, as we have done in gnowsis.

A URI can be used to identify an abstract concept.

Again we have an example, the identification of the concept of “love”. A solution here would be to use WordNet[10] identifiers for the meaning of English words, as Dan Brickley suggested in [Brickley2001]. According to his definition (and a correction by Libby Miller [Miller2003]), “love” could be expressed with this URI.

```
“http://xmlns.com/wordnet/1.6/love-4”.
```

Some common concepts can be identified with this method, Dan Brickley and Libby Miller used it to identify the concept of a “person” in their FOAF, project. [FOAF].

For many concepts, we need another approach. What is a good identifier for the “XML Lecture” from our own example? From the student’s point of view, it means attending the lecture and all related actions, like doing the exam or writing the homework. Restricting the focus on digital resources, the student has to organize his resources that are needed for reaching his goal, doing the exam. A precise definition of this concept would be: *The term* XML Lecture *is my participation at the lecture and is used to classify my digital resources*.

This concept needs a URI to identify it and to relate resources to it. This URI has to be subjective to one student. It has no meaning for other students attending the same lecture, as they have their own definition of *I participate at the lecture*. A resource is needed that can be used as a representation of the concept. It is not useful to have URIs that does not identify a resource, even the “Love” URI returns a RDF/XML document when it is entered in a web browser. Using a real resource has also the advantage that it is unique; a random URI may already be used by something else. Any common resource like a file, folder or email could be used to represent a concept. As an example, the file-folder for the “XML Lecture”, can be used as a class to identify the subjective “XML Lecture” concept of the student. The folder is already defined as containing resources that are needed for the lecture. Its URI can be reused to classify the emails, appointments and persons that are related to the lecture.

Since the URI crisis is an unsolved problem we chose that a URI identifies all four different things that it may identify: name, concept, location and document. Resources can be used to represent abstract concepts; it is the responsibility of the user to pick good resources that represent the concepts in his mind. This approach is used as workaround, until the W3C publishes a recommendation about this topic.

In the following section, the practical implications of this approach are discussed.

3.5.3 Grounding of a concept

An interesting question remains how a URI and a resource correlate to the thoughts and concepts in the mind of the user. From the user’s point of view, it is important to correlate a thought in the mind to a resource in the computer. First there is a thought in the user’s mind that leads to the decision to search for a resource in a computer; then the user has to find this resource that his thought is related to.

[10] http://www.cogsci.princeton.edu/~wn/

If a user thinks about “My participation at the XML Lecture at the Vienna University of Technology”, there are different resources in his computer that correlate to this lecture and it is not clear which resource the user will like to see.

- The webpage of the lecture at the Vienna University of Technology may be stored as a bookmark in the browser of the user.
- The user always searches at google to find Clemens Kerer and XML.
- The date and time of the lecture are noted in the date book, titled “XML Lecture”
- There is an address book entry for Clemens Kerer
- The script of the lecture, written by Clemens Kerer, is stored on the harddisk in a folder named “XML Lecture”.

The thought is like an electrical current that originates in the user’s mind and searches for a place to discharge, a resource that matches the thought. So if the desire is to find a place where to “discharge” the “thought”, it needs a grounding. We define the grounding resource as follows:

For a thought in the mind of the user there may exist a grounding resource. It is the resource that is associated best to the thought. The symbolic representation of a thought is the grounding resource.

The term *grounding* was also mentioned in Georg Dorffner’s book “Konnektionismus”. Concepts and symbols can be designed in a way that they can be experienced in states of a computer system. He noted that the term was used in the community of artificial intelligence and knowledge theory researchers. [Dorffner1991, page vi].

The reason to speak of a grounding is that the stream of thoughts continues to flow when it has reached the grounding, now the user can follow the links in the gnowsis system in addition to the associations in the mind.

There can be more than one grounding resource for a thought, all resources in the “XML Lecture” example can function as a grounding. If there is more than one grounding resource, they should be connected by links. But there usually exists one *main* grounding resource that has the strongest association with the thought of the user. Data in the gnowsis system should be linked to this main grounding resource. In the XML Lecture example, the file folder would be the main grounding resource.

As each human person has different associations in his mind and also his computer contains different resources, the principle of the grounding is *subjective to the use*, it describes how one user associates thoughts to resources on his own computer. This model is not exhaustive and has to be discussed further, but it served well during the design of the information management part of this thesis.

3.5.4 Discontinuity of URLs

Files in a file system can be renamed or deleted. If a resource is moved from one host computer to another, the URI changes. So URIs are subject to change and there is no guarantee that the identifier of resource remains constant. If the identifier for a resource changes, triples that relate to this resource will contain “dangling links”. The gnowsis system cannot handle the discontinuous nature of URLs.

The gnowsis system is not designed to provide a solution for this problem, although there are some approaches. If a resource is renamed, either a relation to the new name can be entered using the OWL “same as” relation or all references to the resource can be renamed using a search-and-replace algorithm. Both approaches have their drawbacks and only cure the symptoms but not the disease itself.

A better approach is to create unique identifiers for resources. An example for a file storage system is Lifestreams by Eric Freeman and David Gelerneter [Free96]. In this system all files are named according to a timestamp and conventional filenames are not needed, there are other methods to find the files. The Web approach for constant identifiers is the URN standard as described in [RFC2141]. The approach of the URN was evaluated. A URN is a unique name for a resource, that does not contain its location. A system of distributed servers is queried to map a URN to a locator. The system is not in popular use and the problem is just shifted: What if the URN changes? As it was not really needed for the gnowsis, we chose to exclude the URN idea.

3.6 Idea management through Wiki and Weblog

The gnowsis is an information management system and therefore has to support basic idea management and diary features. As the user interface aspect of this work is not the main focus, we decided not to program a exhaustive interface but to reuse existing technology, preferably free open source tools.

Therefore, the protégé-2000 ontology and information editor [Protégé] was evaluated. It is a Semantic Web application that supports the entering of ontologies and data. Protégé was not chosen because it does not integrate well to the RDF storage server also the user interface of protégé is hard to learn and the RDF data structures created have some disadvantages.

During the evaluation of information management tools, many WiKi / Weblog systems were tested by the author and the WiKi way of managing information is sufficient for the gnowsis system. The combination of WiKi and Weblog, a so called “bloki” was chosen for the gnowsis. After an evaluation of several Java based systems, the decision has been made to use SnipSnap by Fraunhofer FIRST [SnipSnap] as information management and diary application for gnowsis.

3.7 User Interface GnoGno

The user interface to gnowsis is a separate software project that is based upon the gnowsis server and may be exchanged with other systems. This user interface is called GnoGno and is responsible for enabling the user to work with the data stored in his gnowsis. The name GnoGno was created as a friendly sounding counterpart to the “gnowsis”. It is a typical Wiki name.

Creating links between resources and following these links is the main purpose of the GnoGno user interface. The problem with creating links is that the resources that have to be linked are in different hosting applications. When starting a link, the first resource may be the person “Clemens Kerer” in an address book. The second resource may be a folder labelled “XML Lecture” where some related work is stored. The user interface to create the link has to be integrated into the address book and the file explorer application. We decided to place the linking user interface in a third application, the “Linker”. To link any resource, the hosting application has to provide a button labelled “link” and call a function of the GnoGno system. When the second resource is found, the second hosting application also provides a button “link” and the GnoGno is informed. The GnoGno-Linker itself is a separate stay-on-top window that is shown when a linking process starts and is closed when the linking ends. By this separate window, it was easy to implement linking functionality on a system-wide level.

A similar approach is used to follow links to other resources. If the user searches for resources connected to a given resource, the hosting application of the resource has to provide a user interface for this. Programming this interface for all different hosting applications is expensive, so the following of links is also programmed with a separate application, the GnoGno-Browser. In the hosting application, a button “browse” is added that enables the user to browse for resources that are related to an active resource. The GnoGno-Browser appears and the user can see what resources are connected and navigate to them.

By centralizing the link and browse functionality, to integrate an application it is sufficient to add two buttons and pass a URI to a function. If the user interface is not needed, any third party application can interact with the data in the gnowsis directly and add links in its own way. Also the navigation to other resources is implemented as a function of gnowsis and so any resource can be opened just by passing its URI to the system.

The browser and linker programs have to be exchangeable, as other researchers might create better user interfaces for them. In GnoGno, these programs are loaded dynamically into the system and can be exchanged.

Browsing through all information in detail is also an interesting feature. It is realised by adapting the SnipSnap WiKi system and creating a website where all information can be browsed by the user.

3.7.1 User Context

It is interesting to know which resources the user accesses. By knowing what a user has accessed, applications have the possibility to infer on what problem the user works and how the user is solving this problem. Based on this information, applications can support the user during computer interaction [Schwarz2003].

The gnowsis system has to support a basic user context system, because of two reasons. First, there was a need for good default values where the user has to select a resource from a set of choices. Recently used resources were a good idea. But more important was the fact that through the URI identifiers, it was possible to describe the user context not only in textual terms but also in globally unique context URIs, which are a better basis for future work on this topic.

3.8 Concise Bounded Resource Descriptions

To display information about a resource, first a representative selection of the information is needed. If a file is shown in a file list, typically the name, size and last change date are shown, this is a good selection to represent a file. There is no clear boundary where a resource starts and another resource ends and which information is important to describe a resource and which information has to be omitted. Having all resources connected in a web-like structure, an exhaustive description of one resource would be its place in the whole web. This section is about a standard that can be used to select the data.

Patrick Stickler's URIQA standard was evaluated and his definition of the *Concise Bounded Resource Descriptions (CBRD)* has been adapted and implemented in our gnowsis system [Stickler2003].

As this model is very new and is not common knowledge, it will be described here. The short definition is:

"A concise bounded description of a resource is a body of knowledge about a named resource which does not include any explicit knowledge about any other named resource."

[Stickler2003]

The original URIQA concise bounded description of a resource includes:

- all statements where the resource is subject
- Recursively for all statements already found, if there is an anonymous node, include the statements about it, too
- Recursively for all statements already found, include the CBRD of any reifications.

This should constrain the description to only those statements made explicitly about the resource in question.

Regarding the web-like structure of the gnowsis system and the need to visualize the results, the CBRD definition has been adapted to include some additional useful statements:

- For any resources found, include the `rdf:type` and `rdfs:label`
- Include all statements that have the resource in question as object part.

This changed model of a concise bounded resource description will be called a *bounded description* through this thesis. It is used to select data for display to the user. Selecting the `rdf:type` and `rdfs:label` of all resources also selects data from the predicates that are returned in the triples. So some ontology information is included in every bounded description. An example for a file resource is given in Appendix A.

3.9 Development of the prototype

In the W3C, it is a common practice that new standards are accompanied with prototype implementations[11]. These prototypes have to be open source and are needed for several reasons. First, they prove that the specification can be implemented, which is not always clear. Second, they serve as a correct reference

[11] http://www.w3c.org/Status gives an overview of reference implementations

implementation of the specification. Third, they are license free open-source software, they are the first thing an interested person can try out and use.

Similar reasons implied the creation of a gnowsis prototype. The idea of the architecture had to be tested. During the development, it changed and was improved. At the end of the process, many errors where detected and corrections had to be made, some open issues remain. To have a reference implementation is not important for the gnowsis, as it is no public specification. But there is a need to standardize the interfaces in the future.

Another reason to program the gnowsis prototype was to show how a Semantic Web application may work today. There are only a few true Semantic Web systems available. During the creation of this thesis, some new products were announced on the RDF-Interest group mailing list[12], about one or two in a month. Some of those were released under an open source license, others were commercial products. In the young field of Semantic Web, it is important to have sample applications that show how to use the available tools and how to build applications. So the gnowsis was also designed as a showcase of up to date tools. The gnowsis architecture is documented and can be reused. It is a framework for other information management tools and can be adapted. The three major subsystems are layered and it is possible to completely remove the higher layers. In custom projects, only a subsystem of the gnowsis can be used. The data transformation parts can be used independently from the information management system.

The topic of a Semantic Desktop is important for future computing, as perhaps all software will integrate into a Semantic Desktop. The paradigms of a it will move to mobile devices, a similar system will be needed for mobile phones and Personal Digital Assistants (PDA). There is a need for standardization here, the interfaces between subsystems and the programming interface for developers have to be open standards. Different vendors could then implement a Semantic Desktop Server. Like a SQL database, such a server should be exchangeable. To support this need for standardization, the author based the whole gnowsis system on open source and free software tools. The gnowsis system itself is released under a free license. After the completion of this work, the author will engage in standardizing some aspects.

As the resources of this project are limited, the prototype is not very stable. It had to work in test scenarios, it was not tested exhaustively. Also the user interface is designed in a simple way. The features of the system are still limited. Some of the features identified in the requirements analysis could not be implemented. The system was used by the author for Personal Information Management and proved to be useful. The development will not stop at this level, it is possible that a succeeding project will continue this work.

3.9.1 Tool selection

What programming tools can be used to create a system like the gnowsis and how should the software process look like?

As there exists no budget for gnowsis, all tools had to be open source and without license fees. This lack of money did not decrease quality, as nearly all tools that are available for the Semantic Web are research prototypes or license free.

[12] http://lists.w3.org/Archives/Public/www-rdf-interest/

The system was designed to integrate into the major address book software Microsoft Outlook and therefore the Microsoft Windows platform has been chosen, because the author is an experienced Windows programmer.

Many academic tools are written in Java and also the mature Semantic Web framework *Jena* [Jena] is Java based, so the best programming language for the server was *Java*. Fast user interfaces that have convenient features like drag and drop are best written in a system dependent programming environment like Microsoft C++ or Borland Delphi, but as these tools are commercial, they could not be used. Also, if the Server was already written in Java, the user interface could be integrated easier with Java. To enable integration, some programming interfaces were created. Namely, the gnowsis system publishes its functionality with *Java RMI, ActiveX and JavaBeans*.

Publishing the functionality in an ActiveX Object was done through the JavaBean-ActiveX wrapper by Sun, this wrapper is only shipped with Java 1.3 and 1.4.2. Picking the version of the Java environment was not trivial, we decided to use 1.4.1 because of the URI support that was crucial and the regular expression support. A runtime of Java 1.3 was installed for the ActiveX wrapper.

To cross from Java to ActiveX, the *Jacob* System by Dan Adler was chosen. It is freeware and open source.

The visual information management software was based on *SnipSnap* [SnipSnap]. Also, a web server has been integrated, the *Jetty* server was chosen because it is open source and can be embedded in any application.

For RDF processing, the framework *Jena* by HP laboratories is a good solution that is often used in Semantic Web research systems. Jena supports RDF storage in files and databases and has an exhaustive framework for RDF manipulation, including support for reasoning and ontologies.

Development of the Java applications was done with the *Eclipse* platform, that was originally created by IBM and is now an independent open source project. This tool integrates tightly with Apache *ANT* and *JUnit*, two common Java development tools that were also used in this thesis. To round it up, the Apache logging utility *LOG4J* was also used, instead of the builtin Java 1.4.1 logging.

The Java AWT user interface was built using the free personal edition of Borland's *JBuilder*.

Extracting metadata out of MP3 files was done with the *MP3* class by Jens Vonderheide [MP3File]. Keeping a Java frame always on top in MS-Windows was done with WinAlwaysOnTop from Oliver Pfeiffer.

Because of this long list of different libraries, incompabilities were a daily routine. Nearly all of these tools use the Apache Xalan/Xerces parsers and some depended on different versions. More tools were shipped with the already chosen libraries: the embedded McKoi RDBMS, Apache Crimson and IBM's ICU4J.

An overview of version numbers and addresses is given in Figure 5.

Name Version	Use url
Java SDK	Basis of Programming the system.
1.4.1	http://java.sun.com/j2se/1.4.1/download.html
Java Runtime	ActiveX wrapper for Beans.
1.3.1_09	http://java.sun.com/j2se/1.3/download.html
MS Outlook	Contact Management, basic PIM. Data was extracted from it.
2002 (10.2627)	http://office.microsoft.com/outlook
Jacob	Accessing ActiceX object from normal Java code.
1.7	http://danadler.com/jacob/
SnipSnap	Wiki / Blog Information management, adapted by gnowsis.
0.4.2a	ftp://snipsnap.org/snipsnap/snipsnap-0.4.2a-20030326-src.tgz
Jetty	Embedded Webserver, used for running Servlets.
4.2.8	http://jetty.mortbay.com/jetty/download.html
Jena	Gnowsis integrated into this RDF Framework.
2.0	http://www.hpl.hp.com/semweb/jena2.htm
Eclipse	Java development, this was the main IDE.
2.1.1	http://www.eclipse.org/platform
Ant	Compiling and Buildscripts. Replacement for Make.
1.5	http://ant.apache.org/
JUnit	Unit testing, integration tests.
3.8.1	http://www.junit.org/index.htm
Log4J	Logging messages to the command line or files.
1.2.7	http://jakarta.apache.org/log4j/docs/
JBuilder	IDE to author Swing / AWT windows.
8.0	http://www.borland.com/jbuilder/
Xerces	XML API, used by several tools.
2.2.1	http://xml.apache.org/xerces2-j/
MP3 File	Extract ID3 metadata from MP3 files.
	http://www.rabbitfarm.com/id3.html
WinAlwaysOnTop	Put a Java window in front of other windows.
	http://www.mysrc.net/

Figure 5 Tool Table

3.9.2 Development process

Considering the tool selection, the interested reader may come to the conclusion that the software was created by applying the Extreme Programming technique, but this is not the case. A more conservative software development process was used. Programming turned extreme only at the end of the project.

Building the gnowsis system was done in a controlled software development process. At first, the vision for the system emerged. The first idea of this system was defined in 1996, out of the need for an improved PIM. Development started in March 2002 by collecting ideas and studying the Semantic Web. In October 2002 a rough thesis paper existed together with an abstract. The project was discussed with different external persons. An intense *requirements analysis* was done by the author, it resulted in a requirements and analysis document that described all features of the system in detail. The document consisted of use cases, functional requirements and non-functional requirements and was 41 pages long.

All critical parts of the system have been programmed as *prototypes* during the analysis phase of the project. The prototypes proved that the system could be created using the selected tools and in a reasonable time. The prototypes and the requirements were finished in May 2003.

At the end of the analysis phase, all created prototypes were analysed again and the experience from programming the prototypes was used as a basis to start the system design. The core components were identified using the “post-it” method, were every object is first created as a single paper post-it note and the object’s methods and properties are written on the note. In this project the paper based approach was faster than using a professional tool like Rational Rose for UML design. The post-it objects were grouped and the different sub-system could be identified.

Programming started in August 2003 and ended in October 2003. Because of the existing code of the prototypes and the long-planned architecture, no major problems blocked development and the final system could be built in a relatively short time. It was not possible to start earlier because Jena 2.0 was released in August 2003; the prototypes were built with the beta version.

In the process design patterns were used, especially Adapter, Wrapper and Bridge patterns. [DesignPatterns]

The first release version of the gnowsis is 0.1.

This section explained the basis of the gnowsis and how this basis was formed. In the next section the realisation of these ideas is described, the architecture of the gnowsis prototype.

4 The Gnowsis Architecture

The gnowsis architecture is best represented in a graphical tree model, see Figure 6. It can be compared to a tree, growing on a soil of data and carrying the features like fruits. The basis of this tree is formed by the Adapter Framework. Like roots extracting nutrients from the soil, the adapters extract data from the hosting applications and convert the data into the RDF format. Each hosting application is handled by a separate `AdapterGraph` instance, all adapters are integrated by the Adapter Framework class itself. The Adapter Framework centralizes the roots and builds the stub the server is based on.

The trunk of the tree is the gnowsis server. It publishes the data from the adapter Framework and is the central object in the architecture. The server starts first and creates the different other systems. The server has an additional data source, the RDF repository. It is used to store bidirectional links and other data.

On top of the trunk are the branches and the “fruit” of this tree, the user interface and other interfaces. The user interface is called *GnoGno*. It consists of the `UserContext`, the Browser and the Linker. The functions of the *GnoGno* can be called by other applications.

Applications can use the features of the *GnoGno* and the gnowsis server. The important methods are published through RMI and ActiveX. Using the ActiveX interface, MS-Outlook and MS-Word were enhanced with buttons that call the linker and browser functionality of the gnowsis. When RDF data is requested through the interface, the request is handed over to the roots which extract the data and pass it on back to the branches.

A detailed description of the system architecture follows, with the focus on the technical features of the system and not the biological metaphors. The important algorithms are described. Examples are given as illustration of the use of gnowsis.

4.1 Source, JavaDoc and Prototype

More information about the gnowsis prototype can be found in the source files and the corresponding JavaDoc documentation. The source and documentation of gnowsis can be obtained at the website of this project.

http://www.gnowsis.com

Also a compiled version of the prototype can be downloaded there. It is fully functional and can be used as a reference to the reader. The software is built for the windows platform and support Microsoft Outlook. During installation, some integration and configuration tasks are necessary, these require knowledge about Java.

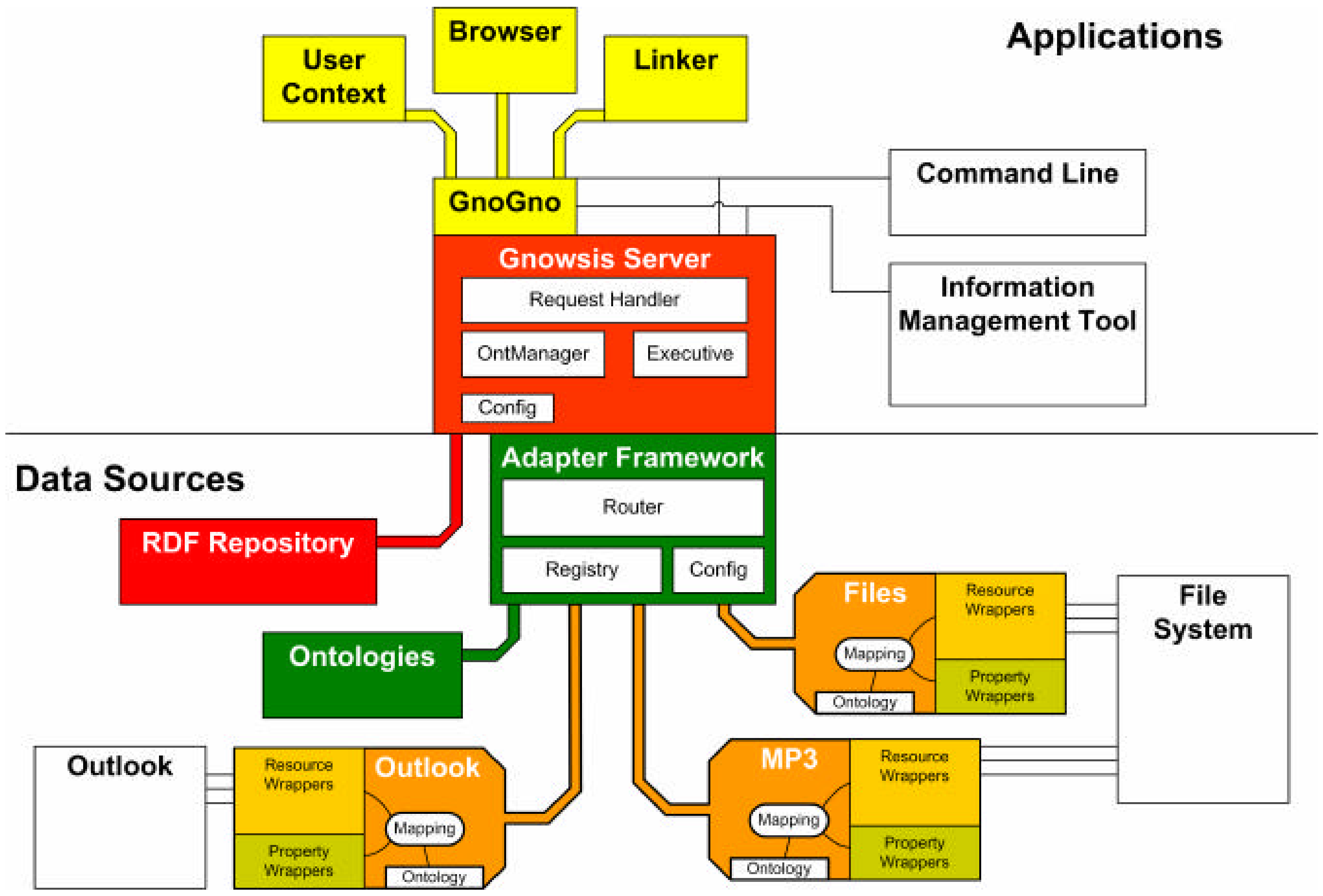


Figure 6 Gnowsis Tree Architecture

4.2 Adapter Framework

The conversion of data from different sources to RDF is achieved by the gnowsis adapter framework. A short introduction to the Jena graph framework and to the extraction process is given. Then the Adapter Framework is described in detail. Each part of the framework is described with its functionality. To show how the parts work together, an example is given how a single request to the framework is handled.

At the end of the framework chapter, the different implemented adapters for Outlook, the file system and MP3 files are described.

4.2.1 File Example

In this section, an example is used to show how the framework works. The file system adapter and its different parts will be described in detail. It is used to read data of the file system via RDF. Two resource classes are adapted: files and folders. The corresponding RDFS classes are `file:File` and `file:Folder`, the are both subclasses of `file:FileSystemEntry`. A folder can contain other folders and files, this is represented with the property `file:contains`. A file has a filename, this is represented with the property `file:filename`. The syntax used here is a usual syntax in RDF applications, `file:File` is an abbreviation for this namespace:

`http://www.gnowsis.org/ont/filesys/0.1#File`.

4.2.2 Jena integration

The result format of the conversion had to be a RDF graph structure, this is a graph with resources and predicates connecting the resources. RDF triples are built of resources and predicates. The used RDF tool *Jena* [Jena] provides all required methods to manipulate graph structures. The triples of RDF data are represented in a *Jena Graph*. The class to manipulate a *Graph* is called a *Model*. A Jena *Model* has methods to add triples and select triples from a Graph. In a Model, the elements of the Graph are represented as Java objects. These objects are Resources, Predicates and Literals and have exhaustive functionality.

When a resource is used in a program, connected resources and literals can be found through methods of the Jena Model. A graph is queried with the URI of the resource as parameter in the find function. The resulting nodes are then represented in lists or collections. With this, the Jena framework allows a convenient way to access and manipulate RDF data.

The triples are stored in the Graph object. The Model just serves as functional interface to the Graph and wraps it. A Graph can store the triples in different ways. The Jena Framework is shipped with different Graph implementations that can store the data in memory, files or in a SQL database like MySql. See “Figure 7 Jena Graph and Model” for a structural view.

Jena Graphs allow the integration of different data sources into the Jena system. We used this feature of Jena to wrap data from hosting applications. The data of hosting applications is made available as normal Jena Graph and can be accessed through a Jena Model to run queries, select triples and browse the data.

To wrap data from an application, one central method had to be implemented, the "`find`" method of the Jena Graph interface.

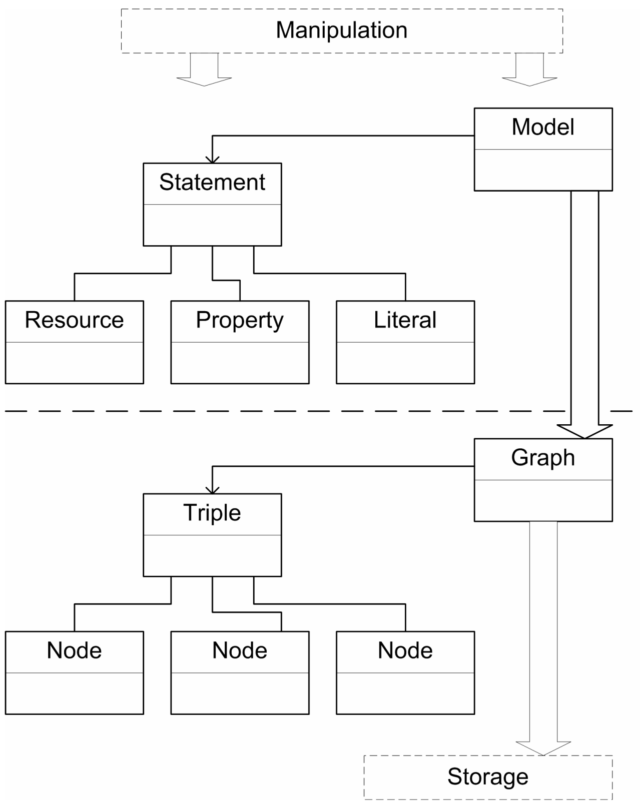


Figure 7 Jena Graph and Model

The `find` method is defined as

```
    public Iterator find(TripleMatch t);
```

The `TripleMatch` consists of subject, predicate and object of a statement. Parts of this triple may be `null` and are treated as wildcards. The graph has to find triples that match the `TripleMatch` parameters and returns the triples in the `Iterator`. The result of the find method is then used by other applications. Normally, a Jena Model accesses the find function, using the results of it to allow access to the graph in a convenient way.

Implementing the graph interface has the advantage that existing Jena functionality can be reused. The Jena RDQL Query language works on the adapted graphs. The data from the adapted hosting applications can be accessed through standard Jena functions.

4.2.3 AdapterGraph

For each hosting application, a separate implementation of the `AdapterGraph` interface was created. A base class called `AdapterGraphImpl` was used as parent of the different implementations. It has the following functions:

- Implement Jena's `find()` in a general way
- Load the `AdapterGraph` configuration from RDF
- Load ontology and mapping files
- Create the `Mapping` object

The functionality of an `AdapterGraph` is distributed through the assisting objects, the `Mapping` and `Wrapper` objects. A detailed description of the assisting objects is given in the next sections. Figure 8 shows the architecture of an `AdapterGraph`.

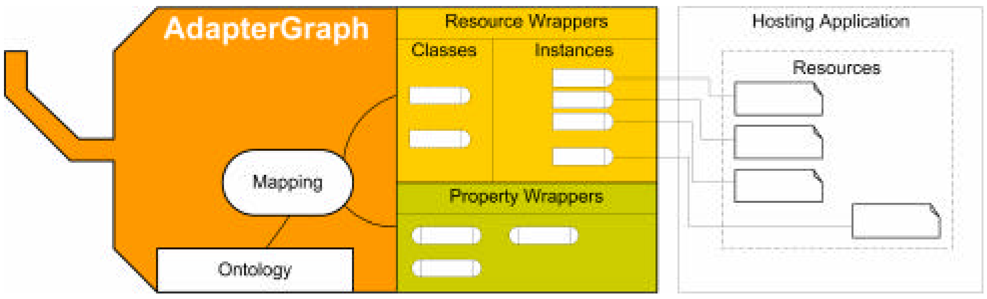


Figure 8 AdapterGraph Overview

The `FileGraph` example is a subclass of `AdapterGraphImpl`. It contains only little because most functionality will be programmed in the resource and property adapters. These are the important methods of the `FileGraph`:

`FileGraph(AdapterConfig item)`
The constructor reads configuration data. The local root path for this `FileGraph` is passed in the `AdapterConfig` configuration. It is important to know the local root folder because files are wrapped relative to this folder and files outside the folder cannot be accessed, for security reasons.

`ResourceWrapper createResourceWrapper(String uri)`
Creates a wrapper for the given URI.

`ResourceWrapper createResourceWrapper(Object resource)`
Creates a wrapper for this object, if it can be wrapped in this graph. This function is used by wrappers that extracted objects from the hosting application and want to wrap them.

`String getGnowsisUrl(java.io.File aFile)`
Get the correct URL of this file that identifies the file in the gnowsis Semantic Desktop. The returned URL contains the hostname and correct path.

`java.net.URI getLocalFileUrl(String gnowsisUrl)`
Transform this gnowsis URL to a path and filename in the local file system. The hostname of the URL is removed and the path translated to the local root.

The class diagram in Figure 9 shows the object hierarchy. The `AdapterGraph` interface on top extends the Jena `Graph` interface. The standard implementation, `AdapterGraphImpl` implements the `find()` method, depending on the `createResource()` method that has to be implemented in a subclass. `FileGraph` implements this method and adds some custom methods, as listed above.

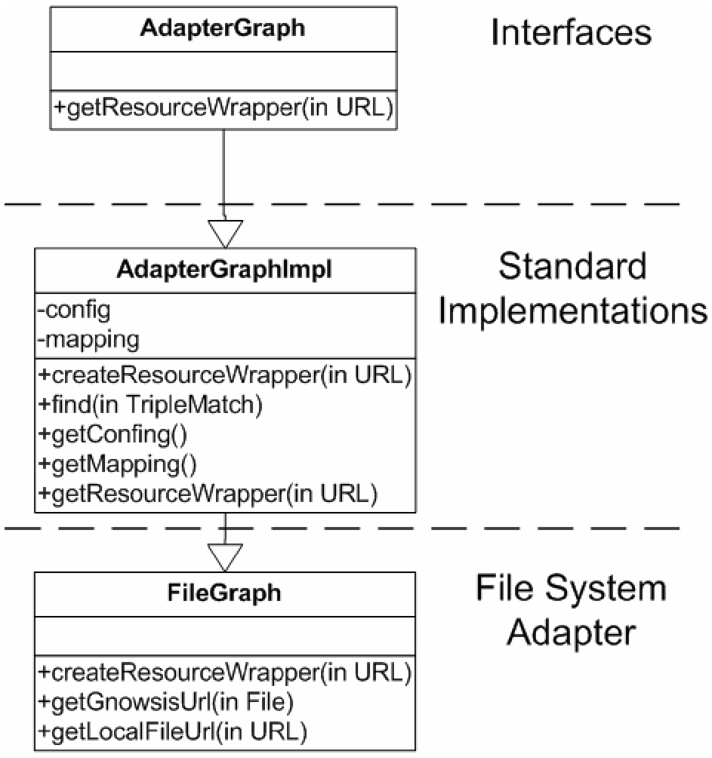


Figure 9 FileGraph Class

4.2.4 Mapping and Ontology

All hosting application adapters are accompanied by an RDFS ontology [RDFS]. It defines which classes and properties are to be expected in the hosting application. The relation between classes and their properties are expressed as RDFS domains and ranges. In the RDF Schema of the `FileGraph` adapter, the classes “Folder” and “File” are defined. The property “filesize” has a domain of “File” and a range of “literal”, meaning that it is a property of files and that the value is not a resource but a literal value (the size expressed as number). For each element a label can be given as a human readable name that can be used in the user interface. Comments on the elements can be included, explaining the meaning of the element as a readable text.

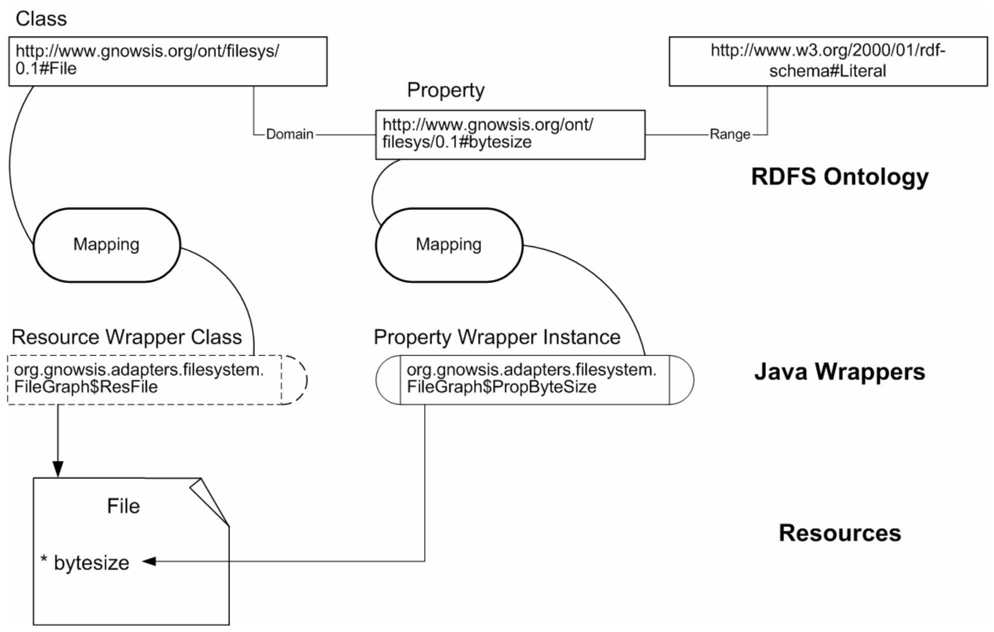

Class
http://www.gnowsis.org/ont/filesys/0.1#File
Domain
Property
http://www.gnowsis.org/ont/filesys/0.1#bytesize
Range
http://www.w3.org/2000/01/rdf-schema#Literal
RDFS Ontology
Mapping
Mapping
Resource Wrapper Class
org.gnowsis.adapters.filesystem.FileGraph$ResFile
Property Wrapper Instance
org.gnowsis.adapters.filesystem.FileGraph$PropByteSize
Java Wrappers
File
* bytesize
Resources

Figure 10 Mapping

The ontology elements from the RDFS ontology are mapped to Java classes. In Figure 10 the ontology elements are represented in the upper area. Each class in the RDFS ontology has a Java `ResourceWrapper` that corresponds to it, this is a 1-1 relationship. In the figure, the mapping is represented by an object connecting the two with arcs. Each property in the ontology is mapped to a `PropertyWrapper` class, the mapping includes for which domain and range the `PropertyWrapper` is configured. In the figure, the Java wrapper "`PropByteSize`" is mapped to the RDFS property above; the mapping implies the domain and range information. Therefore the connection between `ResFile` and `PropByteSize` is taken from the domain of the property and used to map the Java wrappers.

Information about which Wrapper class corresponds to which Ontology element is stored together with the ontology in a RDFS file. The mappings are expressed using a RDFS ontology created by us. This ontology can be found in Appendix B.

The `Mapping` object in the `AdapterGraph` reads the ontology and builds an internal table. This is used to lookup the mappings between the ontology elements and the Java classes. To find which properties a `ResourceWrapper` has, the domain and range entries in the ontology are used. In a triple, the domain is the subject of the triple and the range is the object of the resource.

This is part of the ontology of the FileGraph, in RDF/XML syntax:

```
<rdfs:Class rdf:about='&file;FileSystemEntry' />

<rdfs:Class rdf:about='&file;File'>
 <rdfs:subClassOf rdf:resource='&file;FileSystemEntry'/>
</rdfs:Class>
```

```
<rdfs:Class rdf:about='&file;Folder'>
 <rdfs:subClassOf rdf:resource='&file;FileSystemEntry'/>
</rdfs:Class>

<rdf:Property rdf:about='&file;contains'>
 <rdfs:domain rdf:resource='&file;Folder'/>
 <rdfs:range rdf:resource='&file;FileSystemEntry'/>
</rdf:Property>

<rdf:Property rdf:about='&file;filename'>
 <rdfs:domain rdf:resource='&file;File'/>
 <rdfs:range rdf:resource='&rdfs;Literal'/>
</rdf:Property>
```

To map the items of this ontology, some Java classes exist (the Resource Wrappers) and they are connected to the ontology using the special gnowsis Mapping Ontology. This ontology is defined as RDFS and can be found in Appendix B. This is part of the file that defines the mapping:

```
<rdfs:Class rdf:about='&file;File'>
 <am:hasResMap>
  <am:ResourceMapping>
   <am:resWrapper>
    org.gnowsis.adapters.filesystem.FileGraph$ResFile
  </am:resWrapper>
 </am:hasResMap>
</rdfs:Class>

<rdfs:Class rdf:about='&file;Folder'>
<am:hasResMap>
  <am:ResourceMapping>
   <am:resWrapper>
    org.gnowsis.adapters.filesystem.FileGraph$ResFolder
  </am:resWrapper>
 </am:hasResMap>
</rdfs:Class>

<rdfs:Property rdf:about='&file;contains'>
 <am:hasPropMap>
  <am:PropMap am:useRangeAndDomain='false'>
   <am:propWrapper>
    org.gnowsis.adapters.filesystem.FileGraph$PropContains
   </am:propWrapper>
   <am:hasDomain rdf:resource='&file;Folder' />
  </am:PropMap>
 </am:hasPropMap>
</rdfs:Property>

<rdfs:Property rdf:about='&file;filename'>
 <am:hasPropMap>
  <am:PropMap am:useRangeAndDomain='true'>
   <am:propWrapper>
    org.gnowsis.adapters.filesystem.FileGraph$PropFilename
```

```
      </am:propWrapper>
     </am:PropMap>
    </am:hasPropMap>
   </rdfs:Property>
```

In this example each RDFS-class is mapped to a Java class. The two previously defined properties are wrapped to other Java classes, the classes implement methods to return the triples describing filenames and folder contents. Note the use of `am:useRangeAndDomain`. If this property is set to true, the mapping object will extract the correct resource-wrapper classes using the ontology, f false, the classes can be specified by hand.

The syntax is similar to the normal RDFS syntax, in fact the mapping and RDFS data can be mixed. It is possible to define the `<am:hasResMap>` properties in the ontology file itself. If both ontology and mapping are in the same file, editing the file can be easier.

4.2.5 ResourceWrapper

To access a resource from hosting applications, it is wrapped using a wrapper object. `ResourceWrapper` is an interface and there is an implementing class, `ResourceWrapperImpl` that serves as a superclass for all created `Resource-Wrappers`.

The main feature of a resource wrapper is to contain a pointer or other reference to access the wrapped resource in a convenient way. In the File adapter, all `ResourceWrappers` contain `java.io.File` objects. Every instance also has a URL property that contains a string with the URL identifying the resource.

`ResourceWrappers` are handed to `PropertyWrappers`, these in turn access then the hosting application objects wrapped inside the `ResourceWraper`, so the wrapped objects are usually declared visible to the `PropertyWrapper` implementation (public or package visible).

In our `FileGraph` example, we have two wrappers for the file and folder classes, as defined in the ontology. Folders are wrapped with the `ResFolder` and files with `ResFile`. In the Java SDK, a file and a folder are represented using the same class, `java.io.File`. So both wrappers contain a `java.io.File` object. The only difference is the Boolean flag `isDirectory().` Because the two wrappers are nearly the same, their similarities were expressed in a superclass, the `ResFileSystemEntry`. It contains a `java.io.File` object and functions to map a file to a URI. The two subclasses then needed no functions themselves, besides constructors.

The implementation of `ResFileSystemEntry` consists of functions to return a correct resource wrapper for a URI or a file object. Using `isDirectory()`, a decision is made to create a `ResFile` or a `ResFolder` wrapper.

`ResFileSystemEntry getWrapper(FileGraph graph, String url)`
This function parses the URL and creates a `java.io.File` object for it, this object is then wrapped using either a `ResFile` or a `ResFolder` wrapper.

The class hierarchy of both `ResourceWrapper` and `PropertyWrapper` is shown in Figure 11. In the top section are the interfaces for wrappers. There are standard implementations for the interfaces, these are shown in the middle. The classes of the file system adapter are at the bottom, the resource wrappers to the left and the `PropByteSize` property wrapper to the right. There are other property wrappers (filename, file extension, …) that are not shown in the diagram.

The functions of the property wrappers are described in the next section.

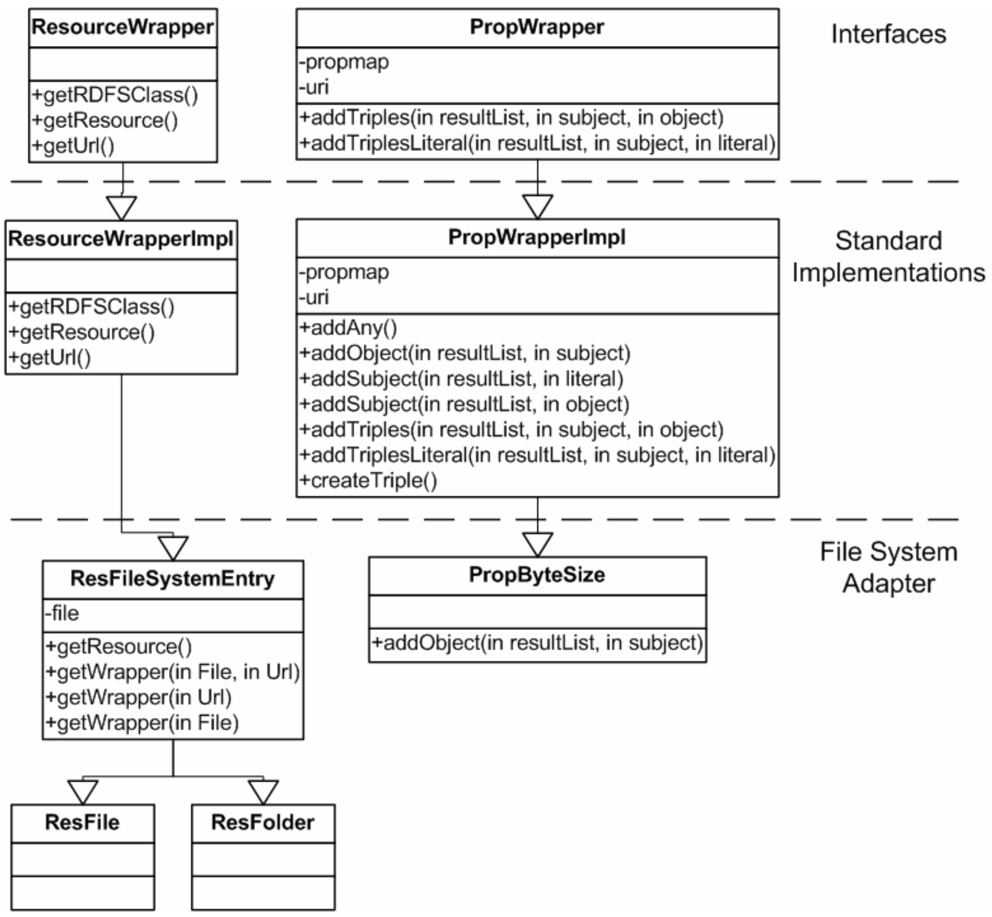

ResourceWrapper
+getRDFSClass()
+getResource()
+getUrl()
PropWrapper
-propmap
-uri
+addTriples(in resultList, in subject, in object)
+addTriplesLiteral(in resultList, in subject, in literal)
Interfaces
ResourceWrapperImpl
+getRDFSClass()
+getResource()
+getUrl()
PropWrapperImpl
-propmap
-uri
+addAny()
+addObject(in resultList, in subject)
+addSubject(in resultList, in literal)
+addSubject(in resultList, in object)
+addTriples(in resultList, in subject, in object)
+addTriplesLiteral(in resultList, in subject, in literal)
+createTriple()
Standard Implementations
ResFileSystemEntry
-file
+getResource()
+getWrapper(in File, in Url)
+getWrapper(in Url)
+getWrapper(in File)
PropByteSize
+addObject(in resultList, in subject)
File System Adapter
ResFile
ResFolder

Figure 11 Wrapper Classes

4.2.6 PropertyWrapper

In the same way that a resource is wrapped in a `ResourceWrapper`, a property of a resource is wrapped in a `PropertyWrapper`. A `PropertyWrapper` has the task to wrap a single property of a resource to a RDFS property. Using an instance of the `PropertyWrapper`, the value of a property can be read and converted to RDF. For example, to access the name of a file, the Java property "Name" is wrapped by a `PropertyWrapper` object and converted to a RDF property. Reading the filename is then done by using the Filename-Wrapper to read the filename of a `ResFile`. The `ResFile` contains a file object that has the wrapped Java property.

Property wrappers have to implement the function `addTriples` and `addTriplesLiteral`, that have a similar header like the find function of Jena. In these functions, triples have to be created that contain the data from passed resource wrappers.

An example:

The resource "XMLLectureScript.pdf" is the script of our XML lecture. Files are wrapped using the wrapper class `ResFile`, which is a child of `ResFile-SystemEntry` and has the `java.io.File` object as a public member. The file resource has the URI

`"file://leo.gnowsis.com/xml/XMLLectureScript.pdf".`

To read the "filename" property, the wrapper instance for the property "`http://www.gnowsis.org/ont/filesys/0.1#filename`" is found in the mapping. The property wrapper is an instance of the class `PropFilename`. Then the `addTriples` method of the property is called, passing the `ResFile` resource wrapper to the property, as a subject in a triple. The property wrapper identifies the missing objects in this triple and accesses the `java.io.File` object wrapped in the passed subject `ResFile`. From the `File` object, the filename can be retrieved using `getName()` and a new triple is then created and returned:

```
Subject:    file://leo.gnowsis.com/ xml/XMLLectureScript.pdf
Predicate:  http://www.gnowsis.org/ont/filesys/0.1#filename
Object:     "XMLLectureScript.pdf"
```

The return format of the `addTriples` method is a list of triples, this gives the flexibility to return more than one triple in a list or add a complete RDF container that consists of multiple triples.

The implementation of `PropFilename` is simple. It can be derived from a convenience class called `PropertyWrapperImpl.` It contains a single method to add the needed triple.

```
class PropFilename extends PropertyWrapperImpl {
 public void addObject(Collection result,
 ResourceWrapper subject) {
  createTriple(
   result,
   subject,
   ((ResFileSystemEntry)subject).file.getName());
 }
}
```

The `createTriple` function is defined in the ancestor, it creates a new triple with a literal value as an object. The triple is added to the result collection. This implementation can retrieve the filename for a given subject. It can simply read the filename from the file. If the search request is in the other direction, getting a subject that suits to a given filename, a more complex function has to be programmed. Such a function was not included. If the request is to return all triples containing the property “filename”, every filename in the system would have to be returned. This was also not implemented.

It is not the goal of the gnowsis adapters to provide full text search functionality or returning all triples of a certain kind. Such searches are best done using the hosting applications themselves. It could be possible to implement efficient search functions to properties, this is left for future releases.

What the adapters can do is extract information from given resources, so that the data is available in RDF and can be used to display or link a given resource.

4.2.7 CommandExecutor

Another feature of the Adapter Framework is the `CommandExecutor` interface. It runs simple operations on the resources in hosting applications. An important operation is to “open” the resource and show it to the user for manipulation.

This is inspired by the “`ShellExecute`” function of Microsoft Windows `ShellApi`. When the right mouse button is used on a file in the Windows Explorer, several operations are available, usually “`open`” and “`print`”. The list of operations is configured in the Windows Registry. `ShellExecute` is then called with the filename of the file and the command to run. By examining the file extension, the correct application is chosen and called with the filename and operation as argument.

The gnowsis `CommandExecutor` is built in a similar way. For a resource, all Adapters are identified that can handle it. Then, each adapter may have a `CommandExecutor` object associated that handles commands on resources. The `CommandExecutor` loads the resource and runs the command on it. Existing functionality of the `AdapterGraph` of the resource can be reused.

A command is identified by a string and passed from the caller to the `Command-Executor`. Developers can define new command identifiers and implement custom commands in a `CommandExecutor`.

It may happen that two command executors are responsible for the same resource. In this case, it is not good to call both executors and have the resource manipulated twice. Instead, a “specific” floating point value is associated with all adapters, indicating how specific the adapter is for the resource, values range from 0 to 10. The file adapter has a low specific, whereas the MP3File adapter has a higher specific. Execution of the command is done with the more specific `CommandExecutor`.

4.2.8 Aggregated Graphs

When the system answers queries, results from several `AdapterGraphs` may have to be combined. This is done with a meta-graph. called `AggregatedGraph`. It is a readonly graph that consists of a collection of other existing source graphs. Requests to the `AggregatedGraph` are forwarded to all aggregated source graphs. The source graphs evaluate the query and return a result in triple form. The results will be collected and returned to the caller as a list of triples. Using the `AggregatedGraph` eases the work with several source graphs, it is a simple way to merge existing graphs.

4.2.9 Router and Registry of Adapters

How to find the right adapter for a resource? At application start, the Framework reads a RDF file that contains configuration data about the adapters. An "*Adapter*" consists of an `AdapterGraph`, an optional `CommandExecutor`, some custom configuration parameters and a set of parameters that define which resources are handled by the adapter. In the configuration files the Java classnames of the `AdapterGraph` and the `CommandExecutor` are listed, the Java classloader is then used to create the objects.

The *Router* object then handles requests. It receives commands from other applications and selects the fitting adapters from the registry. Using the data in the registry, the right `AdapterGraph` for a resource can be found. First, the URI of the resource is parsed and divided into host, path and scheme. A distinction is made between URIs on the localhost and non-local. The URI is parsed. Then, the parts are compared to the configuration of registered Adapters. Every Adapter that matches the requirements may be used to get information about the resource or execute commands. Two adapters may match the same URI, then the data will be merged using an `AggregatedGraph`.

Each Adapter is configured to handle URIs with specific parts. Matching is done according to a configured pattern. Figure 12 lists the available values in a pattern.

Name	Value	Description
onLocalhost	true/false	Is the URI on the local host ?
scheme	string	What scheme is the url (http, ftp, …).
pathStartsWith	String	The path of the URI starts with this string (/ms-outlook).
pathRegEx	regular Expr.	A regular expression that the path has to match.
regEx	regular Expr.	A regular expression the URI has to match.
always	true/false	The adapter matches all URIs. This is used for bidirectional link repository data.

Figure 12 URI Matching Pattern

The Adapter for files in the "My Documents" folder is configured in this way. It contains a `FileGraph` to read the files and a `CommandExecutor` to execute commands. For the Adapter, the pattern shown in Figure 13 was used. For a match, a URI has to conform to all three criteria. The omitted configuration values are ignored (regEx, etc). The string "file://leo.gnowsis.com/data/text.txt" is matched by this pattern.

Name	Value	Description
onLocalhost	"true"	My Documents are local.
scheme	"file"	They follow the file:// scheme.
pathStartsWith	"/data/"	The path starts with data.

Figure 13 My Documents Adapter Configuration

4.2.10 Sequence of Actions during a Request

In this section the sequence of one request is described. This illustrates how the mentioned objects work together. In the following an example is described: a request for all properties of a file resource. The process will be treated in two parts.

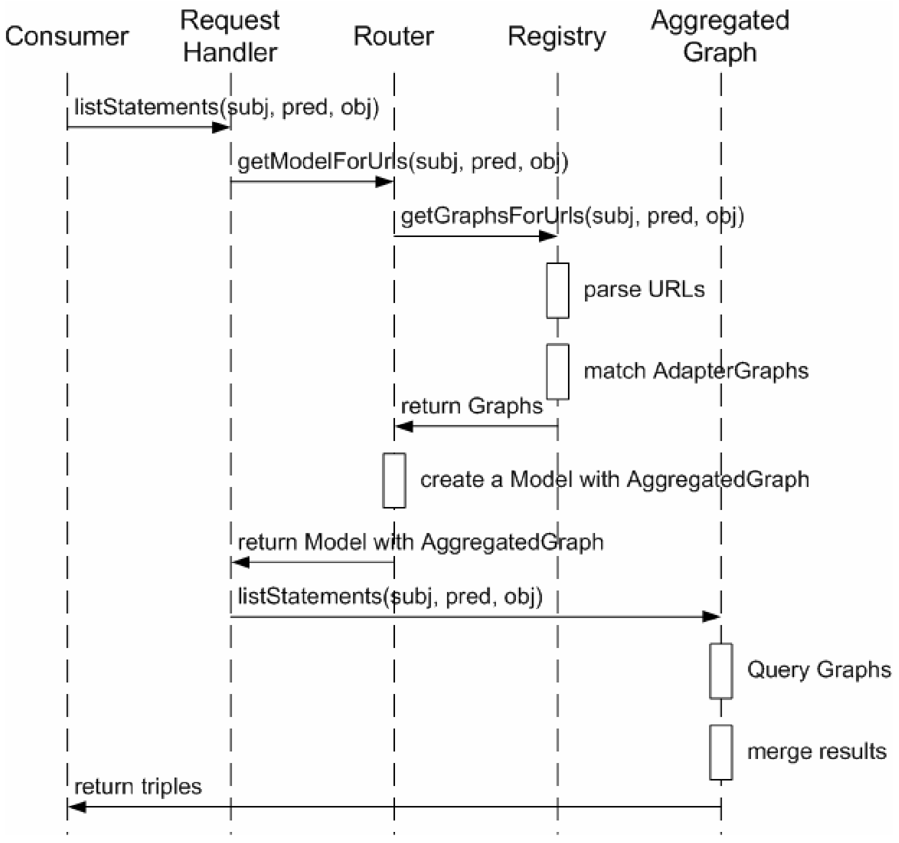

Consumer
Request Handler
Router
Registry
Aggregated Graph
listStatements(subj, pred, obj)
getModelForUrls(subj, pred, obj)
getGraphsForUrls(subj, pred, obj)
parse URLs
match AdapterGraphs
return Graphs
create a Model with AggregatedGraph
return Model with AggregatedGraph
listStatements(subj, pred, obj)
Query Graphs
merge results
return triples

Figure 14 Request Sequence

In the first part, the whole process is described with a focus on the router and registry. In Figure 14 the process is shown in a sequence diagram. A client requests the size of a file. The object `RequestHandler` is the main interface for clients to the server, it will be described in detail in chapter 4.4.6. In this process, the request is rather simple, it is this command:

```
listStatements(“file://leo.gnowsis.com/data/test.mp3”,
“http://www.gnowsis.org/ont/filesys/0.1#bytesize”, null)
```

This means that all triples shall be returned where the given file is subject and the property `file:bytesize` is the predicate. The bytesize is part of the file ontology. First, the contained URIs are extracted from the request, these are the file URI and the predicate URI. The URIs are passed to the router to receive a model that can handle the URIs (`getModelForUrls`). The router passes the request on to the registry. There the URIs are parsed into the corresponding parts (see Figure 12 and Figure 13). These parts are then matched against registered `AdapterGraphs`. Two graphs match the URI. The `FileGraph` of “My

Documents" matches because the scheme is "file", the URI is on localhost and the path starts with "/data". The MP3 graph matches because of the same values as the `FileGraph` and the fact that the URI ends with ".mp3".

The matching graphs are returned in a list (return graphs). Using the graphs, the router builds a Jena Model that contains an `AggregatedGraph` as a data source. This Model can now be used to extract information out of the graphs and is passed back to the `RequestHandler`. The `RequestHandler` now has a Model that can answer to the initial question. The `listStatements` request is run on that Model and therefore on the `AggregatedGraph` in the Model. The contained Graphs are queried and the results merged and returned to the caller.

Once the Model with the `AggregatedGraph` is built, it can be used to serve several requests. These requests have to be about Resources from the same `AdapterGraphs`. For example, further file requests can be run on this Model.

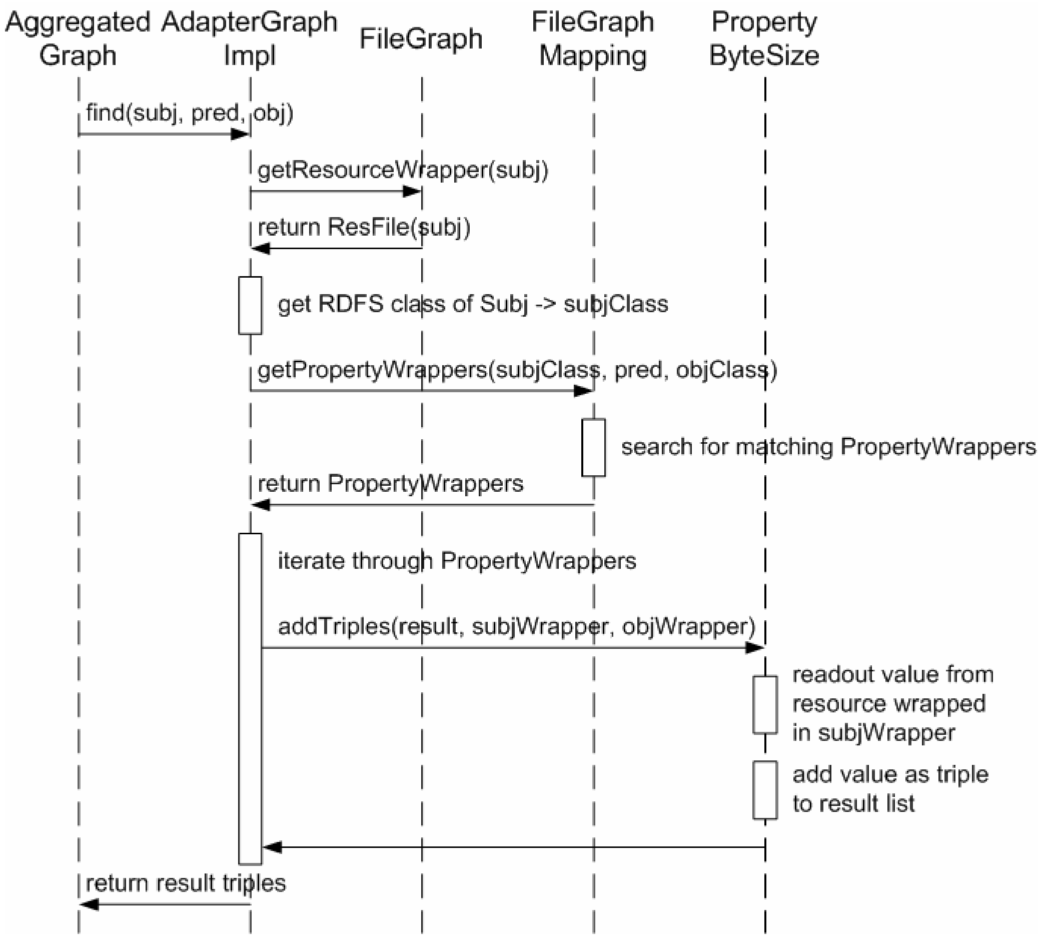

Aggregated Graph
AdapterGraph Impl
FileGraph
FileGraph Mapping
Property ByteSize
find(subj, pred, obj)
getResourceWrapper(subj)
return ResFile(subj)
get RDFS class of Subj -> subjClass
getPropertyWrappers(subjClass, pred, objClass)
search for matching PropertyWrappers
return PropertyWrappers
iterate through PropertyWrappers
addTriples(result, subjWrapper, objWrapper)
readout value from resource wrapped in subjWrapper
add value as triple to result list
return result triples

Figure 15 AdapterGraph find() Sequence

The second part during a request for data is the extraction of triples from an `AdapterGraph`. This is done in the `find()` method. It has to search for triples that match a passed pattern. In our example, the pattern would consist of a file URI and the bytesize predicate identifier, leaving the object null. The find method has to produce a triple with the missing object value.

The find method is implemented in a general way in the `AdapterGraphImpl` object, it is reused by all descendants including the `FileGraph`. Using the generic method `getResourceWrapper(URI)`, the `ResourcWrapper` for the file is retrieved from the `FileGraph`. Then the RDFS class of the returned wrapper is retrieved, this can be read from the mapping in the ontology. As the RDFS class of the subject is now known, fitting predicates can be extracted from the RDF-Schema. In our example, the predicate is already known, it is the bytesize. For this property, the corresponding `PropertyWrapper` is returned by the `FileGraphMapping`. The find method now has a `ResourceWrapper` and a list of `PropertyWrappers` that know how to answer the query. An iteration is made through all `PropertyWrappers`, to get the results. In our case this is only one wrapper for the `ByteSize` property. It is called with `addTriples()`, passing the `FileWrapper` as subject. The `ByteSize-PropertyWrapper` reads the value from the file resource and adds a triple to the result. This triple contains of the file URI, the bytesize identification URI and a literal value containing the size in bytes.

Looking back at the beginning of this example, we see that the URI identifies a MP3 file. So two `AdapterGraphs` are responsible for the file and can extract information out of it, the `FileGraph` and the `MP3Graph`. These are kept in an `AggregatedGraph` to handle the requests. A call to the find method with the bytesize parameter returns a result only with the `FileGraph`. When the `MP3Graph` is used to extract bytesize, it cannot find the property in its mappings and returns no triples. So it is possible to run any requests on any `AdapterGraph`, the request will not fail but only take computation time and the result will be empty.

All requests to extract RDF from the adapter framework follow this sequence. First, the right Jena Model is created to extract data from it and then the wrappers are used to create the RDF triples.

4.2.11 Possible Enhancements

The existing adapter works in the way we described above. An application queries information about a resource. By parsing the URI of the resource, a set of adapters can be identified that may know something about the resource. These adapters are combined into an aggregated graph allowing queries to be made on this graph.

It is possible to enhance the framework. When triples about a resource are queried, connected resources from other adapter graphs may be returned. The underlying aggregated graph does not include the adapter for the new resource and cannot return information for it.

For example, if information about an email message X is queried, the framework returns an `AggregatedGraph` that consist of the Outlook adapter and the gnowsis repository. From email X a triple points to file Y. Using the given `AggregatedGraph`, only the URI of Y is known but no other information about Y can be extracted. To display a link to Y, it would be useful to display the `rdfs:label` of Y, but the label information is in the `FileGraph` of Y. A second query to the adapter framework is needed to retrieve the `FileGraph` that knows about Y.

This situation arose during programming of the Browser application. When a list of connected resources was collected, all resources from the same hosting application had correct labels and types but resources from other hosting applications were limited to the URI.

A solution for this problem is to shift the layers in the Adapter Framework, resulting in an alternative sequence of operations. The result is a *Unified Adapter Graph*. From the outside, all wrapped resources are in the same Jena Graph. All calls to the Jena method `find(TripleMatch)` are handled in a single graph.

For each resource URI, adapters are identified that can handle the URI, this is done with the already used matching algorithm. The adapters which are able to handle the URI are then queried using their `createResourceWrapper( String URI)` function, each adapter returns a `ResourceWrapper`.

From the resource wrappers, the RDFS-classes can be obtained and the needed properties matched, the `PropertyWrappers` are retrieved from the Mapping `Registry`. For each needed property, the `addTriple` method is called with the corresponding `ResourceWrapper` as parameter.

Using this approach, programming could be simpler but the cost to compute the results increases. A buffering system will be needed to buffer the matching `ResourceWrappers` for a URI.

Another good approach to improve the Adapter Framework would be to use the `EnhGraph` interface Jena provides. This has been suggested to the author by the Jena developers in [Jena1]. After publication of the first version of gnowsis, we will contact the Jena developers again about this issue. Updating the adapter framework to support the enhanced graph interface may improve performance and conforms better to the Jena standards.

4.3 Adapter Implementations

Using the Adapter-Framework, implementing the different adapters was straight forward. After one reference implementation existed (the `FileGraph`), it could be used to create the other adapters. Before an adapter can be created, some test application has to be written that shows how to extract metadata from the hosting application. During the prototype tests at the beginning of the project, the data extraction from the hosting applications was already implemented and code from the tests was reused in the final adapters.

First, a Java Package was created, with a unique package name exclusive for the adapter. In this package, a subclass of `AdapterGraphImpl` was created. The graph object can be used to open a connection to the hosting application, like a database connection or an ActiveX to contact Outlook.

The second step is to define the RDF Schema of the Hosting Application. This file contains a RDFS description of the classes that are available in the hosting application and the properties the classes have. This file is saved as “`ontology.rdfs`” into the package. It can be read by the Java environment as a resource.

The third step is to create `ResourceWrapper` objects for all classes in the ontology. For each class defined in the ontology, a `ResourceWrapper` has to be created. As these wrappers may be very similar, it is a good practice to create a superclass for all implementations. The task of a `ResourceWrapper` is to give access to a resource in a hosting application, so the wrappers will contain some pointer or reference to the wrapped resource.

In the fourth step, the missing `AdapterGraph` code is written. Based on a passed URI, the right `ResourceWrapper` has to be identified and created. This involves accessing the hosting applications and creating `ResourceWrappers`.

The fifth step are the `PropertyWrappers`. They access the properties of the wrapped resources. For each property, a separate wrapper has to be written. Again, this task is eased by the use of intermediate superclasses.

The sixth step is defining the mappings of the ontology and the Wrapper classes. The mapping can be either stored in a filed named “mapping.rdfs” (in the package) or directly into the ontology.

The last step is to make all functions work, especially in a multithreaded environment. The extraction process can be similar for different properties, it is possible to write generic wrappers and configure them in the mapping file, we used this approach for the many Outlook properties. A generic string extraction object was written and reused for many properties of Outlook resources.

We implemented four adapters in this way, the key features are described here.

4.3.1 Microsoft Outlook – OLGraph

The Outlook wrapper was designed to extract data from Microsoft Outlook. The actual program version was “2002”. The adapter can work with older and newer versions of the software, as the interfaces have only slightly changed.

The major Outlook classes were wrapped with `ResourceWrappers`, these are:

- Appointment
- Contact
- DistList
- Document
- Folder
- Mail
- Note
- Task

Accessing these objects was done through the COM interface of Outlook. The objects are available through COM and ActiveX, there is an exhaustive documentation about the interfaces in the Outlook Visual Basic help. The important properties of these objects could be easily extracted through the ActiveX interface.

Some properties were not available through the COM interface because of the security restrictions of Outlook. Email viruses have been exploiting the automation objects, so email addresses needed special care. The implementation of some properties was not possible in the duration of the project (f.e. members of mailing lists and their email addresses). Using commercial bridges to access Outlook from Java, these problems are easier to solve. The system works with Outlook in the stand-alone version, it is not useful for a system connected to a MS-Exchange server.

A generic `PropertyWrapper` class was written. It can extract string properties from Outlook objects by their property-name, the `DispName`. This wrapper is configured in the mapping file with the `DispName`. It was possible to extract most data using this generic class.

4.3.2 Filesystem – FileGraph

Files and folders from the file system are available in this graph. There are two resource wrapper subclasses, `ResFile` and `ResFolder`. In Java, both a file and a folder are represented using the same `java.io.File`. Both classes wrap such a file object.

The properties filesize, filename and file-extension were wrapped in `PropertyWrappers`. A property defining that a folder has items was also created.

4.3.3 MP3 ID3 metadata – MP3Graph

This adapter extracts ID3 metadata tags from MP3 files. This standard defines how to describe artist, title, year and other data about the file. The information is then stored into the file. A open source tool has been used to extract this metadata from MP3 files, MP3File by Jens Vonderheide [MP3File]. The common properties of the ID3 standard are available as RDF data.

The `MP3Graph` is configured to be used in combination with a `FileGraph`. If a `FileGraph` contains a MP3 file, it adds the MP3 metadata to the existing file data.

4.3.4 SnipSnap server – SnipGraph

As the information management tool of gnowsis is based on SnipSnap [SnipSnap], a small part from the SnipSnap metadata is also available through an adapter. SnipSnap has a database with wiki and weblog entries, these are used for information management. From these, a readable label is extracted and a distinction is made between weblog and wiki entries.

Each entry in SnipSnap is already referenced by a URI, so the native identifiers can be used in the `SnipGraph`.

4.4 Gnowsis Server

To use the features of the Adapter Framework, a server is needed that can be contacted to run queries. The *gnowsis server* is this central piece in the architecture. In the tree architecture (Figure 6), it is the trunk of the tree. Growing on the roots of the adapter framework it is used by the applications in the treetop.

The *gnowsis server* is started as first object and has references to the other parts of the whole system. It starts the Adapter Framwork and handles the repository. The repository is used to store bidirectional links and other RDF data. All important functions are available through RMI and ActiveX interfaces. RDF information from the Adapter Framework can be extracted.

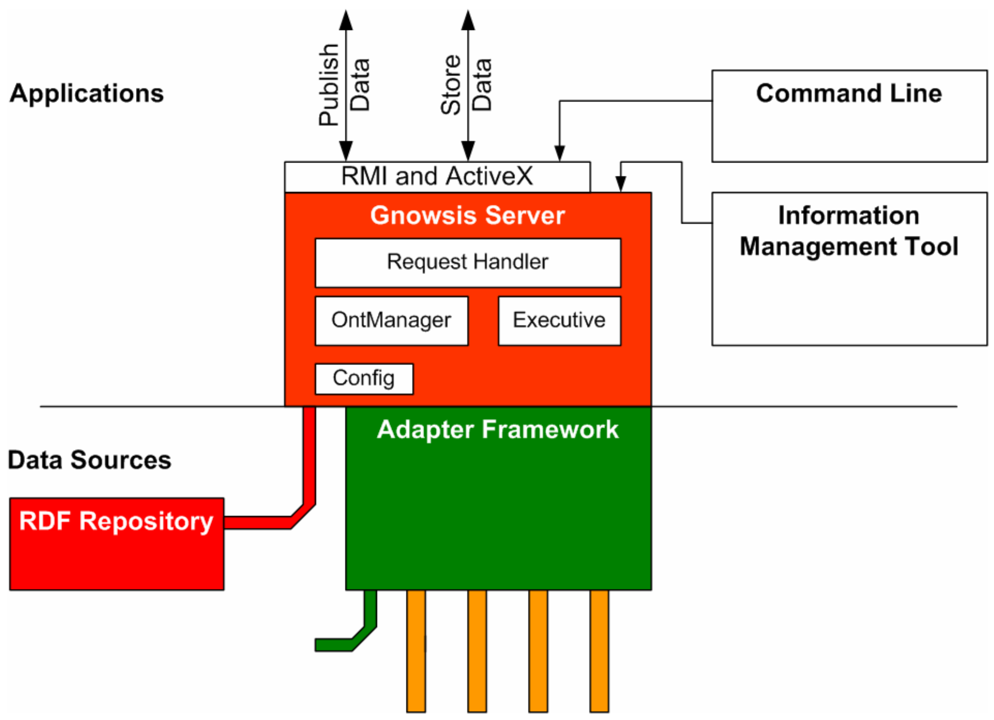


Figure 16 Gnowsis Server Overview

4.4.1 Server Engine

The central object is the server; it has references to all other parts of the gnowsis system and handles creation and initialization of the other parts. Also finalization is done by the Server. The server engine conforms to the Singleton design pattern [DesignPatterns].

Applications that run inside the same process can use the server object to get references to all other objects.

4.4.2 Configuration

The Server needs to be configured. Configuration variables are

- the filename of the repository
- the identity of the user

To handle these variables, a separate object is used, the *configuration*. It loads the configuration data from a RDF/XML file that conforms to a special RDF-Schema.

4.4.3 Repository

The bidirectional links, custom metadata and custom ontologies are stored in this central RDF database. The storage system itself is a file-based Jena Model, all data is stored as RDF/XML serialization in a text file. The file is loaded at startup and stored during shutdown. Using Jena database models, it would be possible to change the file-based system against a database-backed repository, for example mySQL or BerkeleyDB.

The repository is also added as a data source to the Adapter Framework, configured to be included in all queries to the framework. So the data in the repository is always used when data from hosting applications is retrieved, bringing the bidirectional links together with the resources.

4.4.4 Executive

Complex manipulations that are done in the server internally are implemented in this class. The `Executive` serves as a container for often needed functions that are used internally. The main part of it are the *Tagging API* calls.

4.4.5 Tagging API

Applications may use the gnowsis server to store RDF data about resources. This could be data that is about a certain resource or that was created in a resource. For example, the list of participants in the XML lecture is stored in an idea management software and the RDF representation of the list should be available in the repository. To do this, the list has to be stored and marked with an identifier, for future updates and deletion. So we have a *source resource*, that is the list of members. It is stored in some hosting application. The data in the source resource can be expressed as RDF and it is possible to store it in the server. This is needed when the data contains bidirectional links.

When changes are made to the *source resource*, it is not easy to update the RDF representation of the resource. For example, when a new member is added to the participants list, a new triple has to be inserted in the repository. It is safer to delete and rebuild all triples of the resource than changing the single triple.

The triples from one source resource are tagged with the URI of this source, the *sourceID*. A source is usually the resource that was used to enter the metadata, but the *sourceID* can also be an identifier without a real resource. Add, update and delete methods are implemented that take a sourceID as parameter. Data from a source can be added, updated and deleted. Storing of the sourceID is done in a *DataTag*, this can be used to add other metadata like author or creation-date of the triples.

The tagging of the triples was done using the RDF *reification* standard. This seemed to be a good idea and in public discussions this is an often used example of using reification. A disadvantage was that the Jena way to stream reified triples does also multiply the storage space in the chosen RDF/XML file format. Reified triples need about 4 times the space of non reified triples. In database models this problem is not existent because database storage models use row identifiers for reification. The use of reification is a good argument to change from a file based to a database repository.

4.4.6 RequestHandler

This class gathers all important methods for gnowsis programmers. Methods of the Adapter Framework and the repository are accessed through the `RequestHandler`. Design of this class was done according to the Façade Design Pattern as suggested in [DesignPatterns].

As all important methods are implemented here, the RMI and ActiveX interface classes were created as a public representation of the `RequestHandler`. As this class is one of the most important classes in the gnowsis, its methods will be described here in detail.

```
public void storeTagged(com.hp.hpl.jena.rdf.model.Model data,
DataTag tag) throws InvalidTagException, GnowsisException
```

Stores the given data in the repository. All statements are reified and marked as beeing added with the given "tag". The data model has to be a temporary `MemModel` and is consumed and closed during this.

```
public boolean deleteTagged(String SourceIdentifier) throws
InvalidTagException, GnowsisException
```

Delete all triples that where tagged with the given `sourceIdentifier`. Also deletes the corresponding `dataTag`.

```
public void updateTagged(com.hp.hpl.jena.rdf.model.Model
data, DataTag tag) throws InvalidTagException,
GnowsisException
```

Update data that was previously added. The old data is identified by the `SourceIdentifier` of the tag. This method is here for convenience, it calls `deleteTagged` first and then adds the data with `storeTagged`. The passed model “data” is consumed and closed.

```
public String createDataTagId()
```

Creates a new identifier for a new data tag. This returns an URL on this server that is prefixed with a special path for `datatags` and includes a unique number at the end. The number is increased afterwards.

```
public void store(com.hp.hpl.jena.rdf.model.Model data)
throws GnowsisException
```

Store all statements of the given model in the repository.

```
public void store(com.hp.hpl.jena.rdf.model.Resource s,
com.hp.hpl.jena.rdf.model.Property p,
com.hp.hpl.jena.rdf.model.RDFNode o)
```

Add a single statement to the repository.

```
public void store(com.hp.hpl.jena.rdf.model.Statement s)
```

Add a single statement to the repository.

```
public void delete(com.hp.hpl.jena.rdf.model.Model data)
throws GnowsisException
```

Delete from the repository all statements that are in the parameter data.

```
public void delete(com.hp.hpl.jena.rdf.model.Statement s)
```

Delete a single statement from the repository.

```
public void execute(String commandUri, String resourceUrl)
throws GnowsisException
```

Executes a command on the URI. In some implementations, the `commandUri` is ignored and the specified resource will be opened in its hosting application.

```
public com.hp.hpl.jena.rdql.QueryResults execQuery(String
queryString) throws GnowsisException
```

Executes a RDQL query on the server. Result is the Jena-internal query result format.

```
public com.hp.hpl.jena.rdf.model.StmtIterator listStatements(
com.hp.hpl.jena.rdf.model.Resource s,
com.hp.hpl.jena.rdf.model.Property p,
com.hp.hpl.jena.rdf.model.RDFNode o)
throws GnowsisException
```

Lists all statements that match the given pattern of resources. This is similar to the Jena `model.listStatements`.

```
public com.hp.hpl.jena.rdf.model.Model
getBoundedDescription(String uri)
throws GnowsisException
```

Get the bounded description of the resource. What exactly this is, is defined in chapter 3.8, above. This will return all triples that contain the given URI and the labels and types of connected resources.

```
public String getServerHostname()
```

Return the hostname of the server.

```
public String getUserIdentity()
```

Return the useridentity URL of the user.

```
public String fileNameToUrl(String filename) throws
GnowsisException
```

Convert a filename from the file system to a gnowsis URL. This is needed by the file-system-integration tools.

```
public com.hp.hpl.jena.rdf.model.Model getOntologyStore()
```

Return a Model that is the combination of adapter framework ontologies and gnowsis ontologies, that contains all ontologies.

These are the most important functions of the server. As you can see, the interface of the server is similar to the interfaces of typical RDF storage servers [Sauermann2003].

4.4.7 ServerAdmin

Administration calls to the server were centralised in this module. As the administration functionality is only accessible through the web interface, there is no RMI version of these functions.

`ServerAdmin` has only two functions: to backup the server to a textfile and restore it from a textfile.

4.4.8 Ontology Manager

The Ontology Manager provides access to all ontologies on the server. It is used by other internal objects to access the ontologies. Custom ontologies can be stored on the server through a web interface. These custom ontologies are tagged using the Tagging API and can be identified and separated in this way. Through this class, ontologies can be added and deleted from the server, also a list of ontology identifiers can be queried.

Other parts of the gnowsis system buffer ontology data, for example the wiki system buffers all known predicate URIs. To know if the ontology store has changed and the ontology data is outdated, the ontology manager keeps a `DateTime` value that indicates when the data was changed the last time.

4.4.9 Interfaces RMI and ActiveX

Accessing the gnowsis server can be done through RMI and ActiveX interfaces. The RMI implementation is made using a RMI registry that runs in the gnowsis process and needs not to be started, it also uses a custom IP port. The RMI interface is a simpler version of the methods of the Request Handler. While the Request Handler works with Jena objects (which cannot be serialized) the RMI interface is restricted to simple Java classes. RDF values are passed through as subject, object and predicate strings or as serialized RDF/XML strings.

The ActiveX interface reuses the RMI interface. A Java-Bean was written, it runs as separate application and accesses the gnowsis through RMI. Then the Java-Bean was wrapped with the generic Bean-ActiveX Bridge that is supplied with Java 1.3.1 or 1.4.2. We used the 1.3.1 runtime version to bridge the bean. This is considered the best way regarding the fact that other ActiveX -Java bridges are commercial and self-programming is too much effort. Programming such a bridge using the Java Native Interface (JNI) could take a long time and is a tedious task.

For Microsoft Outlook integration, the ActiveX-Beans were of great use. The whole Outlook integration consists of 120 lines Visual Basic for Applications code, similar the Microsoft Word integration.

The server system of gnowsis was kept small. The functionality of the whole system is distributed across the Adapter Framework, the server and the user interface, the GnoGno.

4.5 GnoGno User Interface

On top of our gnowsis tree grows the crown, the GnoGno user interface. It is one of the branches of the tree and uses all the resources of the roots and the trunk. The fruits of this tree are the user interface features, they can be picked by different applications and handed on to delight the user.

As the gnowsis is a new approach for data integration, there were complex requirements to the user interface. To create a link from an address book item to a picture file, we had to access the gnowsis from different hosting applications – Outlook and the file system. Each application provided a distinct technology that allowed us to integrate custom code, MS-Outlook and Word have the "Visual Basic for Applications" (VBA) script language. The file system has the File-Explorer utility (explorer.exe), it is the primary user interface for MS-Windows and can be extended. Taken this perspective of integrating different applications, we had to create a very simple programming interface to the gnowsis.

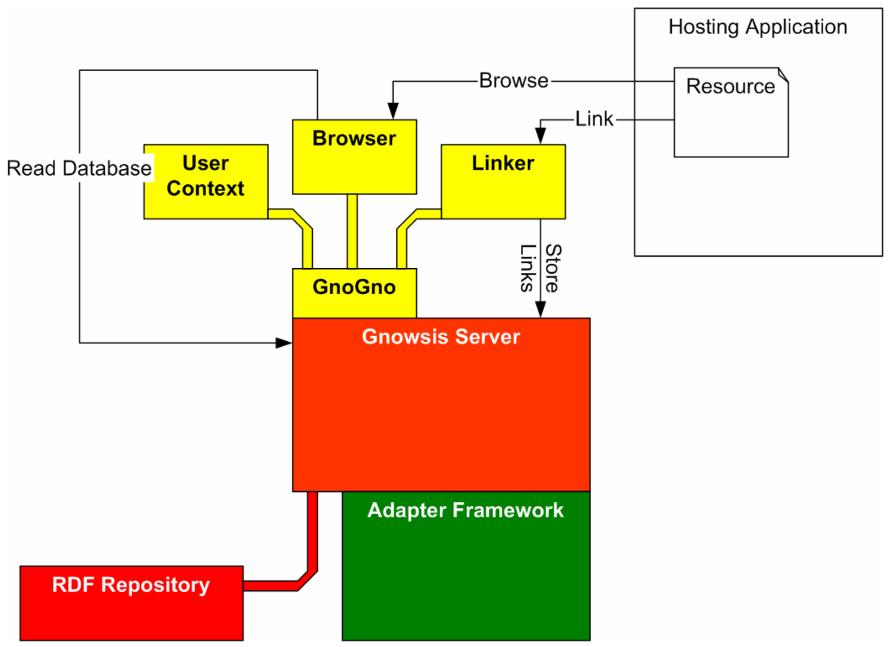


Figure 17 GnoGno Overview

Centralizing the GnoGno functionality helped to build a simple system. Hosting applications or other systems always call a set of functions of the GnoGno. These calls are then forwarded to other applications that implement the functionality.

Three major fields build the core functionality. The *Linker* is responsible to link resources from different hosting applications. Following the links and navigating between resources is possible using the *Browser*. It consists of two parts, a simple Popup-Browser and an exhaustive Web-Based browser. Finally, the *UserContext* listens to all these events and remembers the resources that were accessed by the user.

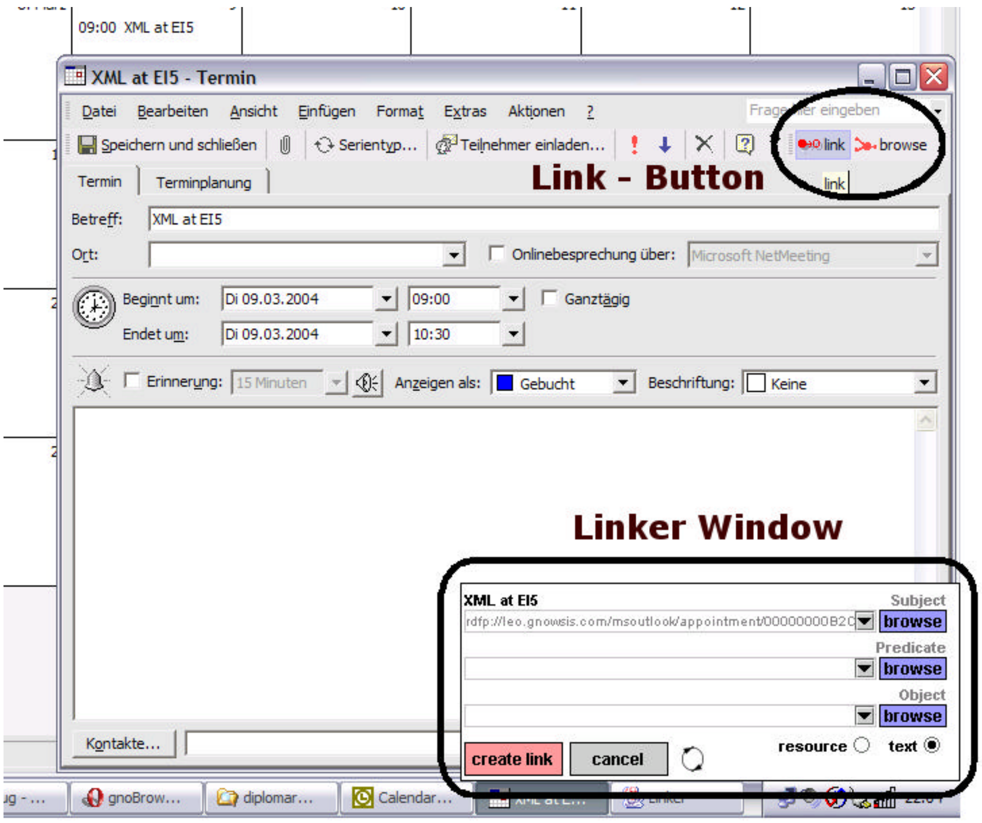
In the first design documents, associating a resource with another was planned to be done this way:

- Open the resource A.
- Press a button to "annotate A".
- A window appears with all existing resources presented in a tree structure.
- Select the second resource from the tree.
- Create the association.

This approach had the problem that all resources had to be displayed in a single tree structure and that this structure had to be optimized to allow the user to find the desired resource in a few seconds. Programming effort would have been high and also the user would have to search through the tree very often. A complete lexical index would have been needed to enable full text searching, which was not planed for the first version of gnowsis. It is better to reuse the search functions of existing applications. By drawing a flow chart of the process of linking resources, another approach was identified that is much simpler and now part of the gnowsis System. The center of our approach is a new object, the *Linker*.

4.5.1 Linker

To link two different resources from different hosting applications, the *Linker* is used. The *Linker* is in the gnowsis system like the clipboard is in MS-Windows: a place to get information from one application to another.



Betreff: XML at EI5

Ort:

Onlinebesprechung über: Microsoft NetMeeting

Beginnt um: Di 09.03.2004 09:00 Ganztägig

Endet um: Di 09.03.2004 10:30

Erinnerung: 15 Minuten Anzeigen als: Gebucht Beschriftung: Keine

XML at EI5

Subject: rdfp://leo.gnowsis.com/msoutlook/appointment/00000000B2C

Predicate:

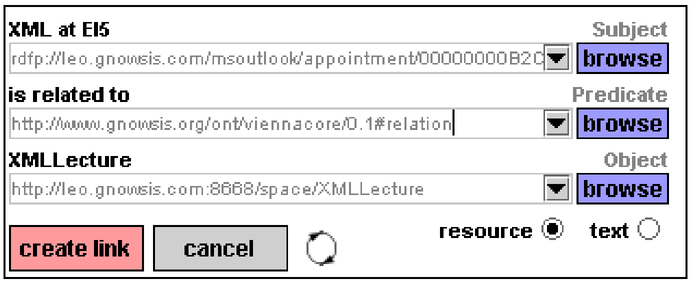
Object:

resource ○ [✓ text ●]

Figure 18 Starting the Linker from Outlook

The user selects resource "A" in application "a" and presses a button "link". The link-button is embedded in this application. Starting the linking process is done and a window pops up and stays on top of the other windows, the *Linker Window*. It contains three fields for subject, predicate and object of a link. Resource "A" is already entered as the subject. In Figure 18, the user pressed the "link" button of an appointment and the linker window appears. Both the button and the window are highlighted with a black border.

Then the user switches to application "b" and selects resource "B" and again presses a "link" button. In the visible Linker Window, resource "B" shows up as object in the link. After selecting a predicate that can connect the two resources, the user can press "create link" and the new link is stored in the repository database. The state just before creating a link is shown in Figure 19.



XML at EI5 — Subject: rdfp://leo.gnowsis.com/msoutlook/appointment/00000000B2C

is related to — Predicate: http://www.gnowsis.org/ont/viennacore/0.1#relation

XMLLecture — Object: http://leo.gnowsis.com:8668/space/XMLLecture

[✓ resource] text

Figure 19 Linker with all needed Data

Alternatively, when the first resource is selected the second resource can be chosen from a dropdown-list of recently used resources (these are taken from the UserContext, see chapter 4.5.4). These recently used resources offer a good choice of default values.

Using the *Linker* from different applications allows the user to browse to the desired resource in any convenient way and then press the link button there. The integration effort is minimal, as only one function has to be implemented in the hosting applications.

Internally, the GnoGno Linker is a class that publishes one major function:

```
public void linkResource(java.lang.String resourceURI)
```

> With this method, a resource is linked. The only parameter is the URI identifier of the resource. The linker window is shown. It is responsible to show the status of the linker to the user.

4.5.2 Linker predicate selection

A speciality of the linker is that the user has to select a predicate that describes the connection between the two resources. The connection may be "they are somehow related" or a specific relation like "this person attended this meeting".

In RDF triples, expressive predicates are used to relate subject and object. Predicates are defined in ontologies. In the FOAF (Friend of a Friend) vocabulary, a predicate exists to describe that a image resource is the photo of a person. A triple that uses this predicate might look like this:

```
    <person> <http://xmlns.com/foaf/0.1/img> <image>
```

As the system knows what RDFS class type the resources are, it was a good idea to include the existing ontologies in the linker. When a user has to select a predicate to connect two resources, the system searches the ontology store for predicates that fit the subject and object resource. If the subject is a person and the object is an image file, the `foaf:img` predicate is a good choice to relate the two. Some generic default predicates like “is somehow related to” are always available as a default choice, also recently used predicates are available. The selection of the predicate is done in the dropdown-combo of the predicate field.

If the student wants to describe that he has a photo of his lecturer, he may link the person of the lecturer to the photo using the `foaf:img` predicate. The process is started by pressing the “link” button at the person. Then the photo is selected for linking. To link a file, the “send to” command of the Windows Explorer was reused. The file system integration is shown in Figure 20 (note that this is a German Windows). Figure 21 shows the linker window with selected person and image.

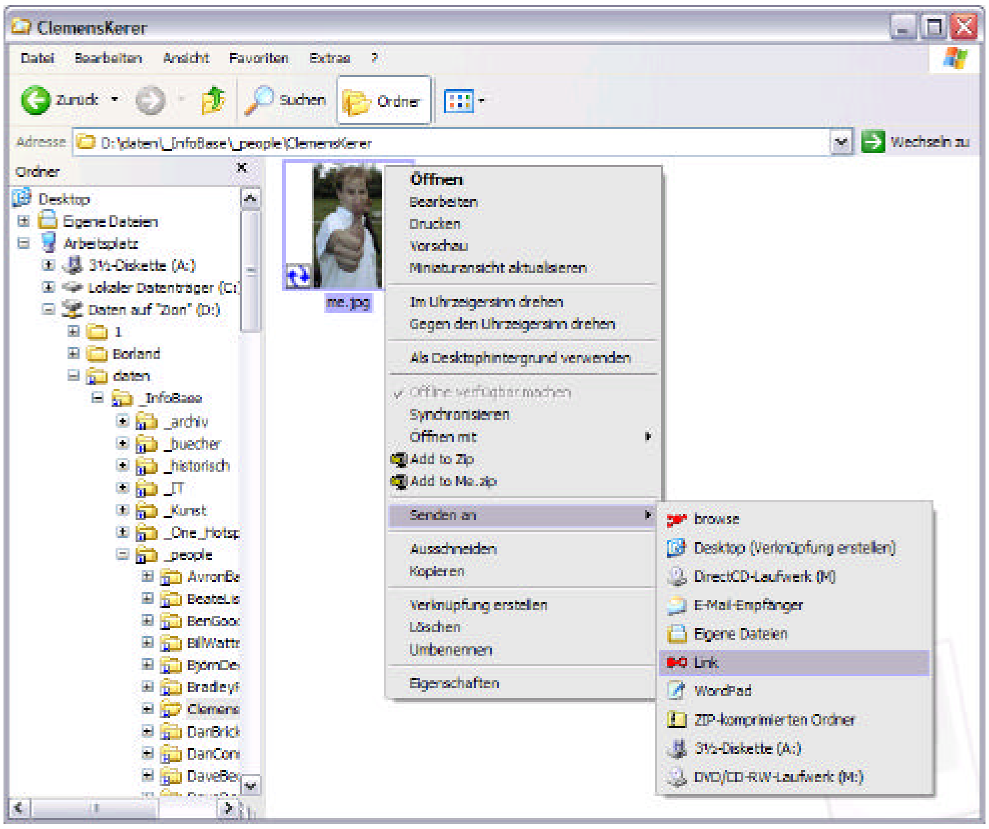


Figure 20 File system Integration

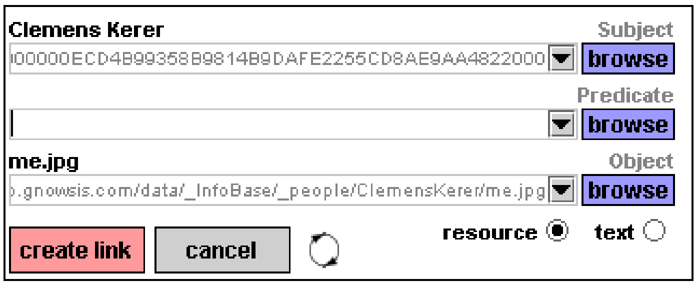

Clemens Kerer
Subject
)00000ECD4B99358B9814B9DAFE2255CD8AE9AA4822000
browse
Predicate
browse
me.jpg
Object
).gnowsis.com/data/_InfoBase/_people/ClemensKerer/me.jpg
browse
create link
cancel
resource ●
text ○

Figure 21 Starting a image link

Then the predicate has to be selected. When the combo-box button is pressed, a Java function is triggered to search for the possible predicates. The user context (see 4.5.4 below) is queried for recently used predicates and the ontology store is queried for standard predicates that fit always. The class of subject and object are analyzed to formulate a special query to the ontology store, for retrieval of predicates that would fit in this triple. This query is run on the known ontologies. When all possible predicates are identified, they are shown in the combo-box dropdown menu. Figure 22 shows how this dropdown box may look, note the third item "image" that is defined by FOAF. The first item "is related to" is good default value for vague relations, the second item is also from the FOAF vocabulary.

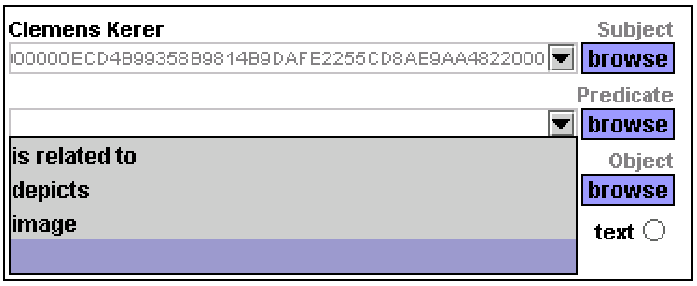


Figure 22 Elements in the predicate dropdown

After selection of "image" from the combo, the URI of the predicate is written into the combo box. The complete link is shown in Figure 23.

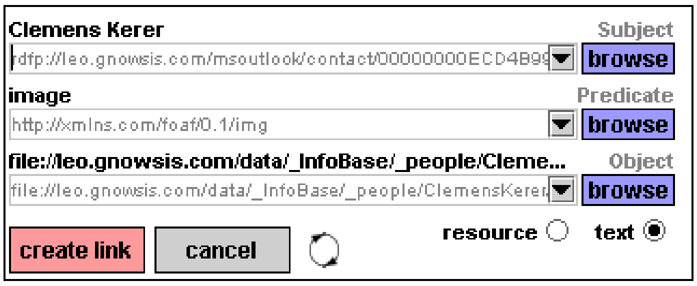

Clemens Kerer
Subject
rdfp://leo.gnowsis.com/msoutlook/contact/00000000ECD4B99
browse
image
Predicate
http://xmlns.com/foaf/0.1/img
browse
file://leo.gnowsis.com/data/_InfoBase/_people/Cleme...
Object
file://leo.gnowsis.com/data/_InfoBase/_people/ClemensKerer
browse
create link
cancel
resource ○
text ●

Figure 23 Complete image triple

This is a practical example of how ontology information can be integrated into daily information work.

4.5.3 Browser

To see what resources are connected to an active resource, the browser is used. The browser visualizes links in the personal information space. It is embedded into hosting applications with a “browse” button. Calling the browser is done with the function `browse(String uri)` that takes a URI as only parameter. This simple function enables the navigation from one resource to another. A popup-window appears which will stay on top of other windows and shows the passed resource together with a tree of connected resources. Connected resources are listed, grouped by the predicate that relates them to the start resource, see Figure 24.

The user can select a connected resources and open it in the corresponding hosting application or keep browsing to other connected resources. As the popup-browser has a limited view, more information about a resource can be accessed through the information management system browser, the *gnoBrowser* (see 4.6 below). It is a website and accessed through any standard web-browser.

Clicking the right mouse button on a resource in the browser opens a context-menu. The choices in the menu are “open”, “link” and “browse”, to open the resource in the hosting application, link it or browser more related resources.

The browser is a quick way to jump from one resource to another. It is a simple visualization of the gnowsis links.

Access to the Browser interfaces is available through RMI and ActiveX interfaces, especially to the `browse` function.

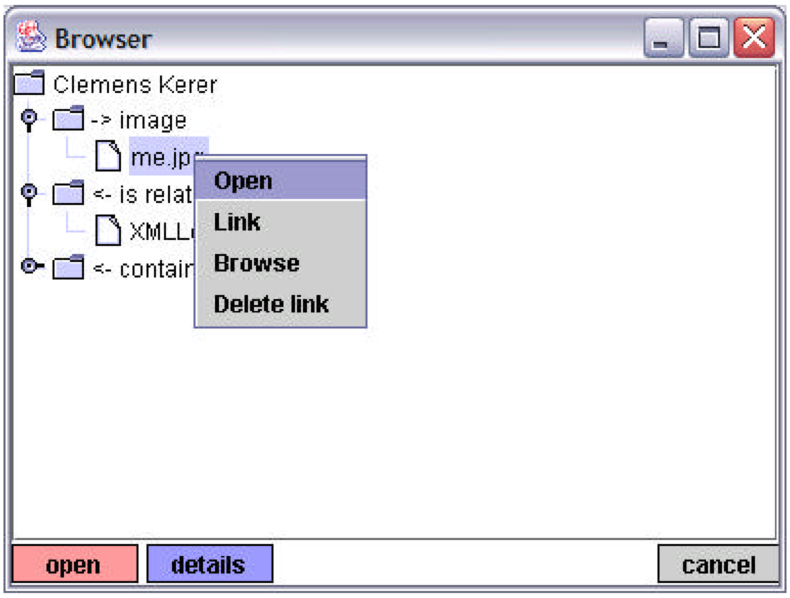


Figure 24 Browser Window

4.5.4 User Context

During longer interaction with a computer, users usually work on tasks. A task can be short like “looking in the calendar” or longer like “writing a scientific paper” or "answer all emails in the inbox". Viewed from a task perspective, we see several tasks that are done parallel or one after another.

The short tasks that involve only one or two resources and the longer tasks involve more resources. We focus on the longer tasks, where several resources are accessed. They are linked together by the nature or topic of the task. Reading all emails in the inbox is connected through the inbox folder. Resources accessed while learning for a university course are connected through the topic of the course and the script.

Accessing the different resources can be tracked by a software system. By calculating what nature the resources are, it is possible to infer the task the user is doing. [Schwarz2003]. Schwarz suggested a Case-Based-Reasoning approach, to determine the user's most probable goals for a given sequence of user actions. We wanted to implement this algorithm but limited by our scarce resources, we chose a simpler approach and model. We have already introduced the principle of the *grounding resource* (3.5.3, above), describing a way to go from the user's mind to resources in the computer. Based on the assumption, that the important thing for the user are the resources that are on the user's mind, we now take the step back and try to infer what is on the user's *short term memory* by analysing the resources the user accessed. The human short term memory can only hold a limited amount of about 5 to 9 items, depending on concentration and duration. This value is defined in [Balzert1996, Page 548] and also in [Haikonen2003, p. 82].

It is impossible to know what a user really thinks, so we concentrate on an educated guess. We use the name *user short term memory* for our approach. By observing recently accessed resources we approximate which computer resources are in the attention of the user at the moment.

This was done in four layers.

- Observation of user actions
- Collecting user actions in a list
- Calculation of the user short term memory
- Usage of the contents of the short term memory.

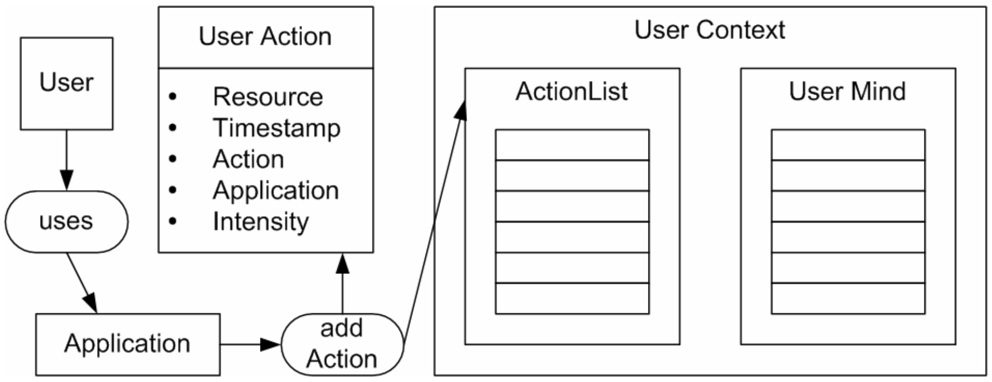


Figure 25 User Context

The whole subsystem in the GnoGno is called *UserContext* and is represented by an object. It contains the two other objects *ActionList* and *UserMind*.

When an application observes that a user accesses a resource, it can notify the `UserContext` by sending a message. The message contains at least the resource, the type of interaction, a timestamp and the attention of interaction, rated as a floating point value from -1 to 1. A positive attention indicates that the resource has been accessed while a negative attention indicates that the resource has been closed or deleted. The messages are not processed on income but buffered in a list, the `ActionList`.

Before it can be used, the `UserContext` has to be updated by calling the `update()` method. This method passes the `ActionList` to the `UserMind` for calculation. In the `UserMind` are objects that represent the items that are in the short term memory at the moment, the `MindItems`. They are calculated based on the `ActionList` using the following algorithm.

```
1. Prune the ActionList downto AMAX elements (removing
   older items)
2. Iterate through all items in the ActionList, for each
   item X do:
   2.1. If there is a MindItem in the UserMind with the
        Resource of X, update this MindItem
   2.2 If not, create a new MindItem with the
       information from X

   For each MindItem M, the M.Importance value is the
   highest X.Attention value that was done on the
   resource. If a negative Attention appears, the
   MindItem is removed.

3. Calculate a "Presence" value for every MindItem M,
   based on the last access time and Attention of the
   UserAction.
4. Order all MindItems by Presence
5. Prune the MindItem list downto MMAX elements
```

The calculation of Presence is done according to this formula (the time values are measured in seconds):

$$\mathrm{M.presence} = \mathrm{M.importance} * \frac{1}{1 + \log\left(\frac{\mathrm{actualtime} - \mathrm{M.timestamp}}{1000}\right)}$$

A useful value for both AMAX and MMAX was 100. It was important to keep older resources in the mind if they did receive a high attention again.

The result is a list of resources, ordered by presence. They resemble the resources that were accessed before. Querying the list is done through the `UserContext` interface and the result of the query can be limited in size through a parameter. Two functions are supported, `getFavouritePredicates` to receive recent predicates and `getFavouriteResources` to receive resources.

This approach is very simplistic compared to the approach of Schwarz and Roth-Berghofer, they are better in detail and provide better results. But it is possible, to exchange the calculation engine and implement the whole model of Schwarz and Roth-Berghofer without changing the interfaces. For us it was important to show a way to a central user context subsystem and how this subsystem may be integrated.

4.6 gnoBrowser

The gnowsis system was created as a basis for future information management tools. The *gnoBrowser* is a prototype for such a new information management system. Visualization of the gnowsis data and integration of this data in text notes are realized.

4.6.1 Information Management by wiki

Implementation of a complete new system from scratch is not a wise idea when existing system can be adapted to be the `gnoBrowser`. We chose a wiki / weblog system, the Java based SnipSnap suite by Stephan J. Schmidt and Matthias L. Jugel [SnipSnap]. The principles of wiki systems are explained above in section 2.5.3.

Wiki systems are to describe separate resources using separate webpages and identifying these resources using wiki-names, using the names themselves as links to the resources. These principles are similar to Semantic Web systems and intensive wiki usage had been an inspiration during the gnowsis development process. So it was interesting to enhance a wiki to a semantic wiki.

4.6.2 Extending SnipSnap

SnipSnap is a wiki that runs in a web server and is accessed through a browser. Based on Java-Servlets it makes intensive use of Java classes and other tools. The Apache standard tag library, custom tag libraries, the embedded Database McKoi and the parser engine *Radeox* (by the SnipSnap developers) are other tools the system uses.

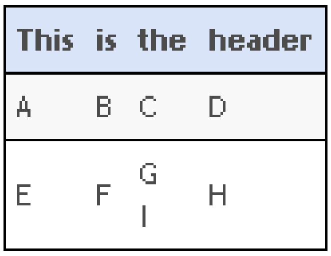
The core item in SnipSnap is a Snip, an item of text information. A Snip is labelled with a unique name or a date. A Snip contains a text. There are some properties, like the last change date etc. The weblog entries and Wiki pages are programmed as Snips. There are servlets to edit, add and delete snips. The snips are stored in a database using the embedded server *McKoi*.

The text content of a Snip can contain special tags that are replaced with text. A tag can be a "macro" or a "filter". A text in curly brackets {} is a macro.

This is a macro to display a label:

```
{table}
This | is | the | header
A | B | C | D
E | F | G \\I | H
{table}
```

It is rendered to this form:



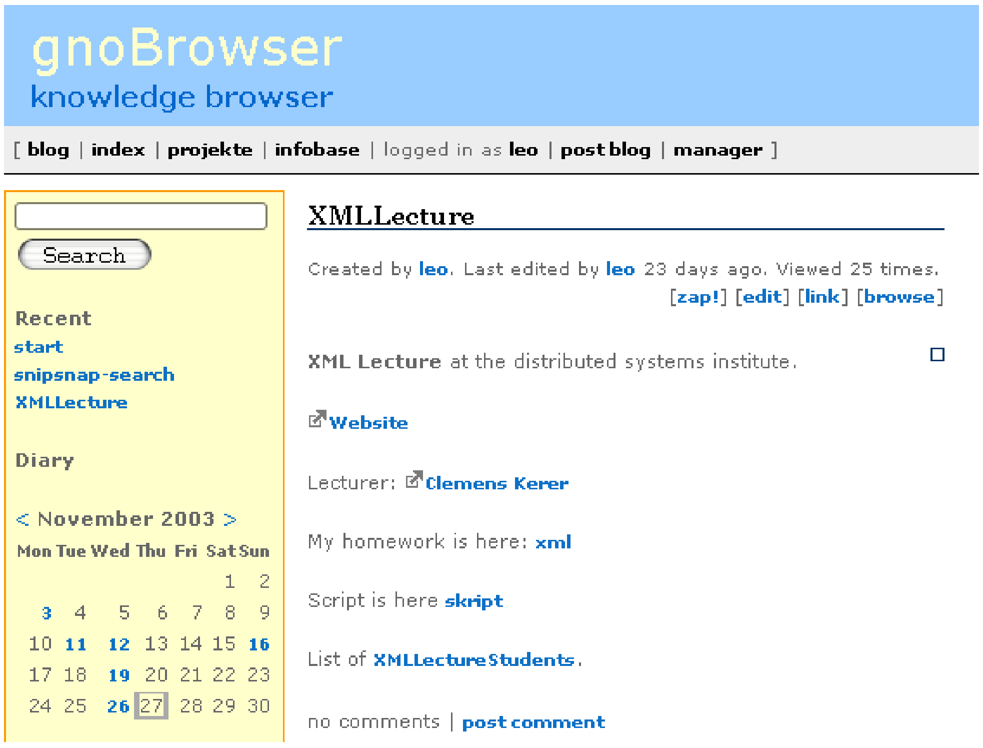
This is the header			
A	B	C	D
E	F	G I	H

A macro can be entered at any place in the text of a snip. Filters are certain characters that are to be replaced with other characters. In above table example, also a filter occurs: the “\\” string is replaced with a HTML `<br/>`. Both filters and macros are implemented as Java classes that can be added to the rendering engine.

Three ways where chosen to extend SnipSnap:

- New Java Server Pages
- New macros
- Hacks into existing SnipSnap code.

4.6.3 Browsing and editing a Wiki entry



gnoBrowser
knowledge browser

[blog | index | projekte | infobase | logged in as leo | post blog | manager]

Search

Recent
start
snipsnap-search
XMLLecture

Diary

< November 2003 >

Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

XMLLecture

Created by leo. Last edited by leo 23 days ago. Viewed 25 times.
[zap!] [edit] [link] [browse]

XML Lecture at the distributed systems institute.

Website

Lecturer: Clemens Kerer

My homework is here: xml

Script is here skript

List of XMLLectureStudents.

no comments | post comment

Figure 26 Wiki XMLLecture

Figure 26 shows a screenshot of a the Snip view. This is a normal wiki page in the system. There are several extension for `gnoBrowser`:

- The link and browse buttons to the top right. They provide the same functionality as in MS-Outlook.
- The links to other resources.
  After the normal text is a region where links to connected resources are shown.

In a normal wiki, it is possible to create links to other wiki pages by entering their WikiWord. In SnipSnap, this is done by entering a word between brackets: "`[WORD]`". A filter will automatically replace these bracket words with `<a href="…">` links to other wiki webpages.

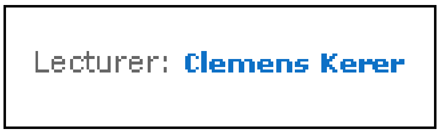
Additionally to this, we introduced another special syntax to link to Semantic Web URIs by putting a URI between angle brackets. Here is the URI of the MS-Outlook entry that corresponds to the lecturer Clemens Kerer.

```
<rdfp://leo.gnowsis.com/msoutlook/contact/00000000ECD4B9
9358B9814B9DAFE2255CD8AE9AA4822000>
```

This syntax will be replaced by the special *Gnowsis Wiki Text Parser*, which is described below in chapter 4.6.5. Instead of the URI, a labeled link will appear in the text. In our example, the link looks like:

```
<a href="http://leo.gnowsis.com:8668/gnowsis/browse?
rdfp://leo.gnowsis.com/msoutlook/contact/00000000ECD4B99
358B9814B9DAFE2255CD8AE9AA4822000">
Clemens Kerer</a>
```

Rendered in a browser, it looks like this. The link is to the right.


Lecturer: Clemens Kerer

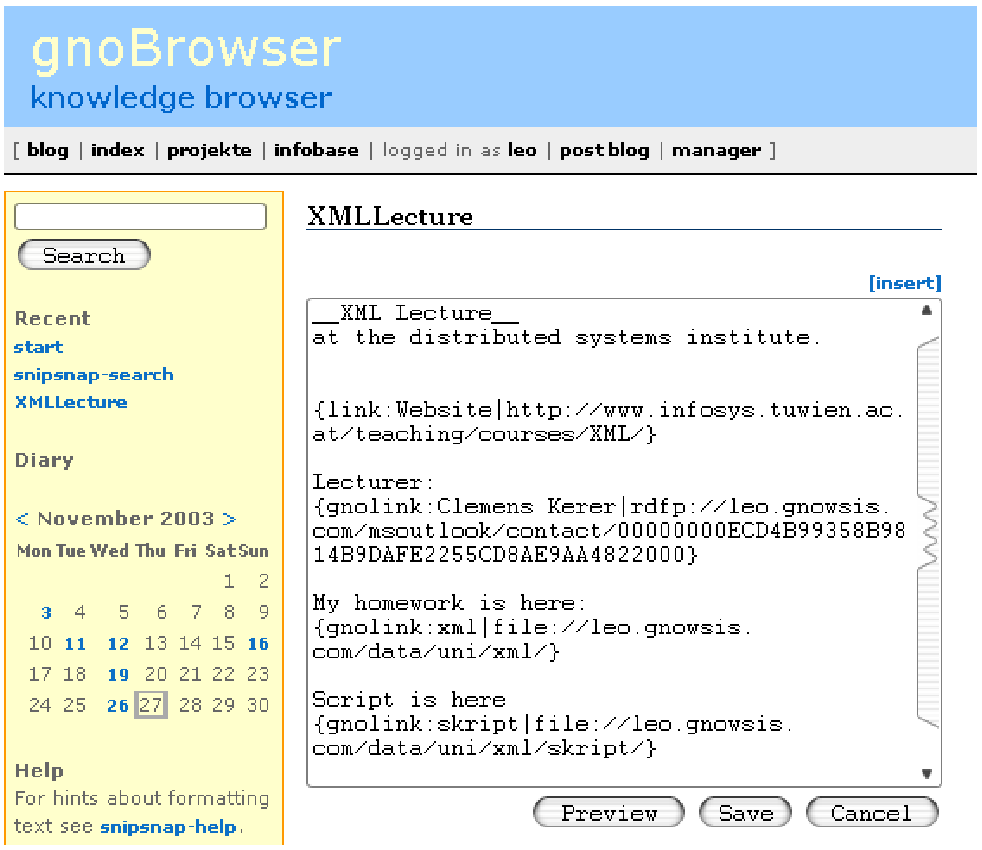
Changing existing information is done by pressing the *edit* link in the top right corner. In the following window, the text content of the snip is changed. In this example we see the entered text and also the text version of the link to Clemens (Figure 27).

In a common wiki links can only point to other wiki pages or websites. The gnowsis wiki was extended to include links to any resource on the system. This enables the user to make use of the resources outside the wiki. We were faced with the task of enabling the user to insert these links into the wiki text. The user interface has to support the selection of a resource in another application and then put the selected resource into the text at hand. As we already embedded the linker and browser applications in the hosting applications, we reused this functionality. Data from the open linker window and recently used resources from the user context can be entered in the web-based wiki.

On top of the text edit field is a link labeled “[insert]”. Clicking this link opens a popup html window where the wanted resources are visible. Recently used resources and the current items of the linker (subject, predicate, object) are listed here as links. Clicking on a link inserts the correct gnowsis URI in the wiki text. It was important to open a new browser popup window to run an up to date query on the server.

To include a link to the instructor of the XML lecture, our student opens the page about the lecture and edits it. A plain text is entered first: “lecturer : “. Then the link to the lecturer has to be added. First the entry of “Clemens Kerer” in Outlook is opened and then the “link” button there is pressed. The linker window shows up and “Clemens Kerer” is now noticed by the gnowsis as being an active resource. In the `gnoBrowser`, “insert” is pressed and the popup window shows, with “Clemens Kerer” listed. The user selects the name and a correct gnowsis link is added to the text content.

More about using links to Semantic Web resources is described in section 4.6.5 below.


gnoBrowser
knowledge browser
[blog | index | projekte | infobase | logged in as leo | post blog | manager]
Search
Recent
start
snipsnap-search
XMLLecture
Diary
< November 2003 >
Mon Tue Wed Thu Fri Sat Sun
Help
For hints about formatting text see snipsnap-help.
XMLLecture
[insert]
Preview
Save
Cancel

```
__XML Lecture__
at the distributed systems institute.


{link:Website|http://www.infosys.tuwien.ac.
at/teaching/courses/XML/}

Lecturer:
{gnolink:Clemens Kerer|rdfp://leo.gnowsis.
com/msoutlook/contact/00000000ECD4B99358B98
14B9DAFE2255CD8AE9AA4822000}

My homework is here:
{gnolink:xml|file://leo.gnowsis.
com/data/uni/xml/}

Script is here
{gnolink:skript|file://leo.gnowsis.
com/data/uni/xml/skript/}
```

Figure 27 Wiki during editing

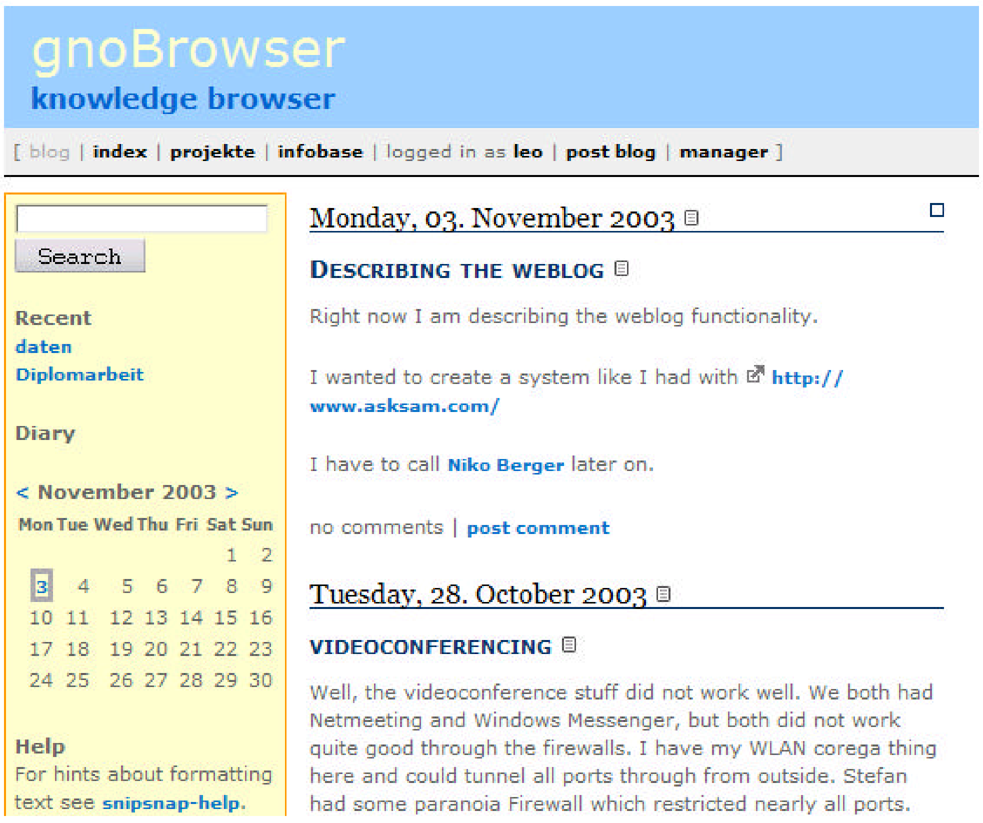
4.6.4 Diary / Weblog

The diary function of gnowsis was built by adapting the existing weblog of SnipSnap. Usually a weblog is published on a website and can be read by every visitor. We provide the weblog on a local web-server, so it is not public but restricted to the user of the local machine. As SnipSnap is a mature Weblog, it provides a comfortable way to post a blog through a web interface.

To select a specific date, a calendar is available in the left menu. In the calendar area, a single month is shown. The dates are highlighted links when a weblog entry was entered for the day. Clicking the date navigates to the page where details about blog entries can be viewed.

A few recent entries are shown in the "blog" section, it is the start page of the system and can also be reached through the "blog" menu. A list of dates and text is shown, the newer entries are in the beginning, see Figure 28.

Pressing "post blog" opens a page to write a new blog entry. A new entry consists of a title and a text, the form to enter this information is simple, as shown in Figure 29. Referencing from a blog entry to another is done by entering a text that conforms to the wiki syntax.



Mon	Tue	Wed	Thu	Fri	Sat	Sun
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

For hints about formatting text see snipsnap-help.

Monday, 03. November 2003

DESCRIBING THE WEBLOG

Right now I am describing the weblog functionality.

I wanted to create a system like I had with http://www.asksam.com/

I have to call Niko Berger later on.

no comments | post comment

Tuesday, 28. October 2003

VIDEOCONFERENCING

Well, the videoconference stuff did not work well. We both had Netmeeting and Windows Messenger, but both did not work quite good through the firewalls. I have my WLAN corega thing here and could tunnel all ports through from outside. Stefan had some paranoia Firewall which restricted nearly all ports.

Figure 28 Weblog start page

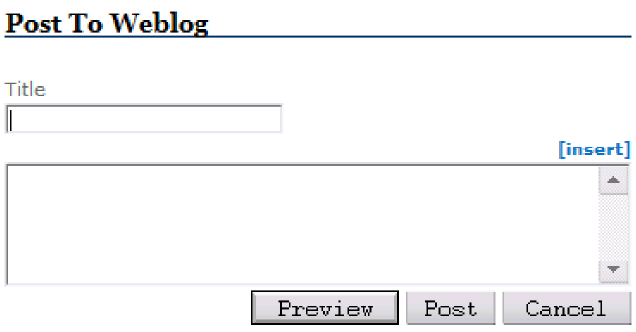


Figure 29 Post to Weblog

4.6.5 Gnowsis WiKi Text parser

Taking a close look at wiki content shows us a chain of plain text rife with links to other resources. Usually some concept is described in the content of a wiki page, the links are embedded to specify what other resources are related to the concept and enable the user to open the resources. There can be different relations between one page and another, analogous to properties in RDF.

In our example, one page is used for a listing of other students that attend to the XML lecture. The list can be used to find students to join for doing homework. In the text of the wiki page, the students are entered as wiki-links, pointing to the Outlook address book entries of the students. When the list is rendered for display, each student's name is a link pointing to an address book entry. Additionally, the list is preceded by the RDF predicate "attends". A link to the XMLLecture is the start of the text, and together with the predicate and the student links, they all form RDF sentences. It is possible to extract the resources and predicates and create a triple representation of the text, the triples can be stored in the repository for global use.

This is the visible text:

Besides me there are other people in the lecture.
XMLLecture is attends by
Matthias Grünberger he will check us an example project ,
Hans - Jürgen Kozik,
and also Christoph Selucky.
We four will perhaps make the assignments together.

Figure 30 Lecture participants - rendered text

The underlined links in the text point to respective gnowsis items. The text is handed to a parser that strips away plain text entries and searches for triples. The parser works in three steps:

- Lexical analysis for gnowsis URIs and delimiters ",.!?", the found symbols are stored in a list.
- Build a symbol tree from the lexical symbols, the tree contains RDF triples.
- Export the triples into a temporal Jena model.

The lexical analysis can identify these resources:

- {gnolink: [<label>"|"] <URI>} – a reference to a URI identifier of a resource
- [<wiki-name>] – a reference to another wiki entry
- {gnothis} – a symbol to represent "this" wiki entry.

The result is a list of symbols, they can be of these types:

- SYM_Resource
- SYM_Property
- SYM_NewSentence
- SYM_SubSentence

The resources are represented with an object of class `SYM_Resource`, that contains the URI of the resource. If an identifier points to a property, it is represented with a `SYM_Property`, to decide if a resource is a property, the ontology manager (see 4.4.8) is used.

The sentence delimiters are “.”, “!” and “?”, the are represented with `SYM_NewSentence` objects. The comma “,” is represented with `SYM_SubSentence`. All symbols identified in this way are collected in a list. Then the list is analyzed to find sentences.

Sentences are of the form:

```
SYM_Resource SYM_Property SYM_Resource SYM_NewSentence
```

The three parts are interpreted as *subject*, *predicate* and *object* of a triple. A sentence may have sub sentences, these share subject and predicate and can contain a new object:

```
SYM_Resource1 SYM_Property2 SYM_Resource3 SYM_SubSentence
SYM_Resource4 SYM_Subsentence
SYM_Property5 SYM_Resource6 SYM_Susentence
SYM_Resource7 SYM_Property8 SYM_Resource9 SYM_NewSentence
```

This is parsed to the following triples (just the numbers):

```
1 2 3
1 2 4
1 5 6
7 8 9
```

The text from Figure 30 is parsed to these triples (items in ankle brackets represent URIs):

```
<XMLLecture> <attends> <Matthias Grünberger>
<XMLLecture> <attends> <Hans - Jürgen Kozik>
<XMLLecture> <attends> <Christoph Selucky>
```

These three students attend the lecture, this information can now be added to the central RDF repository and is available in all queries about the students or the lecture. Also, the students are linked to the lecture and browsing the student will show the relation.

4.6.6 Annotation of any resource

The above examples show how the existing blog and wiki system was enhanced with gnowsis features. It was further enhanced to enter metadata about other resources. The wiki can now be used to annotate a resource from Outlook or a file.

To do this, first the resource is browsed in the popup-browser. There a button “details” will open the resource in the wiki system. There are two extensions to a normal wiki:

- Any resource can be associated with a wiki word
- Any resource can be described using a wiki text.

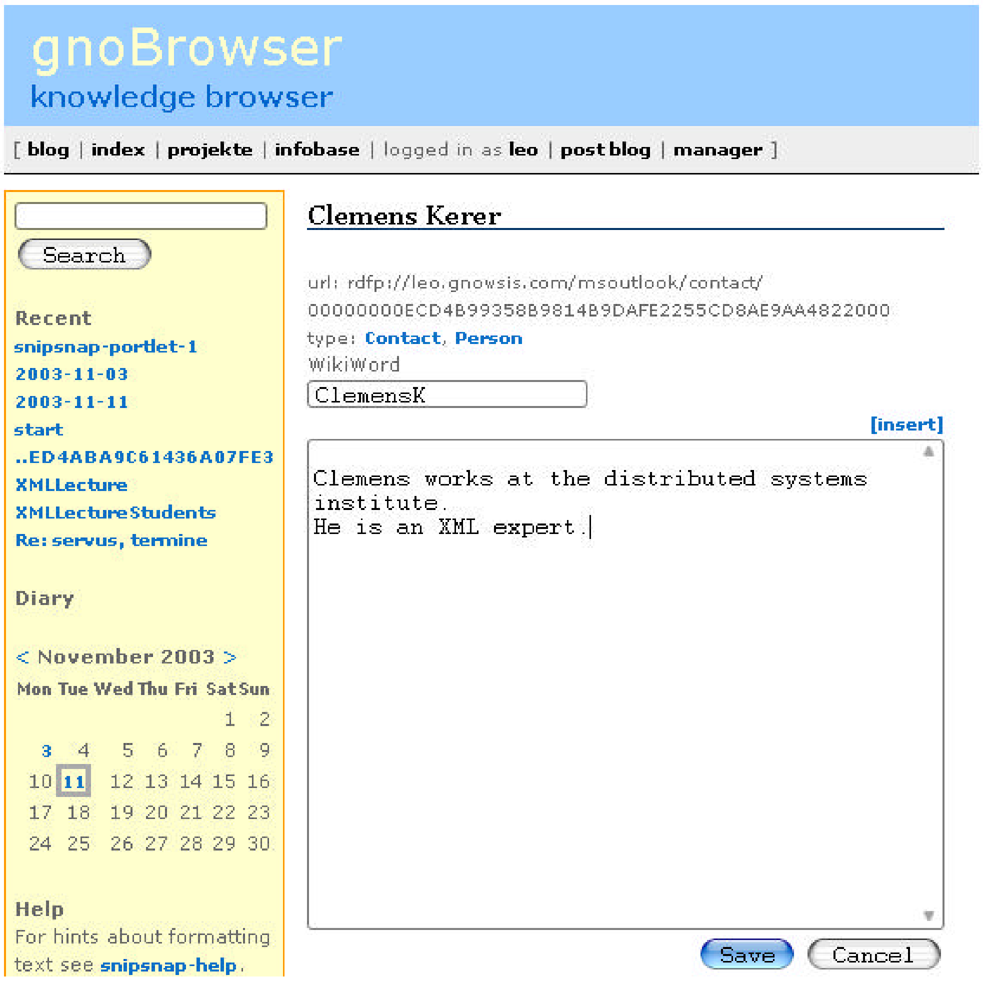
A wiki name can be used as an abbreviation for a resource. When the wiki name is used, the text is linked to the resource represented by the wiki name. Entering common resources in a wiki text is simplified with the abbreviations.

"Clemens Kerer", the lecturer of the XML course, is represented with an address book entry. The URI identifying the resource is

```
rdfp://leo.gnowsis.com/msoutlook/contact/00000000ECD4B99358
B9814B9DAFE2255CD8AE9AA4822000
```

A wiki word is chosen as an abbreviation for this resource, the word is "ClemensK". When this word is used in wiki or blog text, the original resource is associated with it and can be opened.

To describe the resource in detail, a wiki text can be entered. This text extends the functionality of Outlook, as it can contain wiki words and is parsed by the wiki text parser. Figure 31 shows how the annotation of the resource is entered.



Clemens Kerer

url: rdfp://leo.gnowsis.com/msoutlook/contact/
00000000ECD4B99358B9814B9DAFE2255CD8AE9AA4822000
type: Contact, Person
WikiWord
ClemensK

Clemens works at the distributed systems
institute.
He is an XML expert.

Figure 31 Annotation of a Resource

4.7 Summary

The gnowsis architecture shows how RDF can be used throughout a system. Existing resources are identified with URIs and their data is transformed to RDF. Using the adapter objects, it was possible to program three adapters for Microsoft Outlook, the file system and single MP3 files.

On the personal computer, a local gnowsis server publishes the data of the adapters and makes it possible to spin a personal Semantic Web. The local server also published functions to support user interaction. On an embedded web server runs a wiki, enhanced with Semantic Web features.

This system was used to integrate data from different applications and to define bidirectional links between resources.

5 User Experience

In this chapter, the experiences of using the system are described. When the prototype was in a usable state, the author experimented with it to see if the desired features can be used. It was possible to link resources and to browse from resource to resource, but there was no big sensation about it. Using the system became interesting after a starting period, when some resources where already annotated. It is still an interesting question how to classify the resources and what resources to use as classes. This will be discussed in the second part of this chapter.

The test use of the system started in September and ended in November 2003. During this phase, the different parts of the system have been used by the author. A test scenario was writing this thesis. For the thesis, a wiki page was made listing the important resources that are needed to write the thesis. This was also used for management of to-do notes. The XML-Lecture example was entered into the system, to test if the desired features can be used.

Before the system can be used in daily work, it takes a time to get acquainted with it. A learning phase is needed to get over this start barrier.

5.1 Start Barrier

In a fresh gnowsis installation are no custom links by the user. The adapted information from Outlook and the file system is available, but there are no links between files and Outlook resources. The wiki system is empty and also the blog has no entries.

The use of this empty system lacking links is not attractive to the user. Finding information is easier by using the already established classification structure. As the benefit of using the gnowsis is not visible yet, the user will hesitate in creating links and following them.

The first step in using the gnowsis is to enter the most important resources to the *infobase*. Although the data from different applications is integrated, it cannot be found through the `gnoBrowser` if there is no entry point. A root folder is an entry point to the file system. To browse through Outlook data, the contact or inbox folder are good entry points. These entry points are identified through their URIs. The most important project at the moment should also be added to the infobase. Adding these items is done through the wiki text editor in the `gnoBrowser`. The resources are selected with the linker and entered using the insert function. Having the most important resources available in the infobase, it can be used as a portal to the personal data. From the different entry points, links lead to more connected resources.

A good way to get familiar with the gnowsis is the blog. Entering information through the weblog is a very simple process. After pressing the “post blog” link, the user writes what he is doing at the moment. A starting entry could be “I just edited the [Infobase]”. The meaning of this statement is that the content of the infobase was just altered, as suggested above. A link to the infobase is also included in the entry. This is the first bidirectional link.

There are far more links and ideas in the mind of the user, how resources are related to each other and what resources are important etc. Using information management systems like the gnowsis, it is a temptation to classify all resources on the computer. The user wants to build a global hierarchy containing all resources, not caring about future use and the actual need. This is not recommended for the gnowsis. It takes some experience to classify resources and often it is hard to decide which classification something belongs to. At the beginning, only *accessed resources* should be linked. If a resource A is accessed and then the somehow related resource B, this relation is a candidate for creating a link. Especially at the beginning of a new project, it is a good practice to link the resources needed for this project. Working on the project, it is then a useful help to jump from one resource to another using the *browser*.

Grounding resources are special resources that represent a concept. For the beginner, it is not recommended to intentionally create these grounding resources. In our XML-Lecture example, different resources are connected through the idea of the Lecture. The focus is now on the appointment in the Outlook calendar identifying the time and date of the reading and the script file. The script can be linked with a simple "related to" link to the appointment. Both the appointment and the script are grounding resources for the concept of the whole lecture. An experienced user may instead create a wiki entry that identifies the lecture, we have done that for our example. Linking the artificial lecture entry to the reading date is shown in Figure 19. To connect the script to the lecture, a university ontology would be ideal that defines a property "this is the script of a lecture". Entering the data in this precise way is not needed if the intended use is only to browse between a small set of resources.

5.2 Features

A few features of the system were used much more frequently than others. It seems that these are more valued than the other features of the system and so we will describe them in a short way and try to give an explanation of this fact.

5.2.1 Blog

The blog was proven to be a useful feature. From the perspective of a diary, it can be used to express personal feelings about situations, or to write down notes that may be helpful in the future. If an interesting website was visited, a link to the page can be written in the blog – instead of using the bookmarks of the browser. It is sometimes a challenging decision where to store certain information. For example, the news about the publication of a new Semantic Web tool. Shall this information be stored in the file folder "semantic web" or in a research file? In this case the author used the blog to enter the fact. Then the blog entry was linked to a wiki entry about the Semantic Web.

This leads to the perspective of knowledge management. The gnowsis blog can be used as a knowledge management tool. Pieces of information can be entered in the blog and then linked to other resources in the system. A web of information is build in this way.

5.2.2 Wiki

A wiki allows to define every single word that is used in the text and exactly this happens in some public wikis (for example wikipedia[13]). In gnowsis, at first an attempt was made to enter important words into the system and describe them, beginning with things that are important in the author's life at the moment. So the first entry was about this thesis. A link to the thesis document was entered so that the document can be opened from the wiki. Also a wiki entry was created for "Semantic Web things", to relate resources that are about this topic. The thesis was linked to this other entry.

It was a useful feature to note resources that are needed for a project together in a wiki entry. During the test, some restrictions became apparent. The gnowsis is not integrated into important hosting applications like Adobe Acrobat Reader. When a scientific paper is open, the *link* and *browse* buttons are needed to use the features of the gnowsis. Especially to jump from a document to a wiki entry and then further on to related documents.

5.2.3 Linking

All the different resources included in daily information work can now be linked to each other. Emails and wiki pages, appointments and files, contacts and photos are no longer separated but can be connected by links. This feature offers much possibilities. In daily work, the option to connect appointments to files proved to be very useful. If some files will be needed in a meeting, the files can be linked to the appointment entry during the preparation. In the meeting, the files can be quickly found using the browser. This is very useful for Microsoft PowerPoint presentations. If a meeting is a presentation, the needed PowerPoint file can be linked to the appointment. Clicking on the link opens the file and the presentation can start.

Travel preparations can be eased using the linker. The date of the travel in Outlook can be associated to related resources that are needed during the travel, such as the map of the city, the time table of the train and the contact person at the destination.

5.3 Idea of a classification system

As explained in the introduction, humans are gifted with the skill of placing things in a ordering system. Such ordering systems are also represented in computers. In Microsoft Outlook, folders are available to separate resources or categories can be associated to them. It is possible to show items grouped by category, for example in the contacts folder. If we enter a categorization structure in one application, it can appear in other applications, too. The classes "business" and "private" are represented as file folders, as Outlook folders, as Outlook categories, etc. One of the goals of gnowsis is to use the same categories in different applications.

To bring this feature to life, the gnowsis needs to support classes. As RDF is available throughout the system, the use of RDFS classes is in principle possible. The question is, how to enter the classes into the system and how to select them and add resources to the classes. In our XML-Lecture example, we have to create

[13] http://www.wikipedia.org/

the class “these are resources that are related to my participation at the XML Lecture”. As every class is also a resource, we only have to create a resource and define it as being the class, the resource is then the grounding resource of the idea (see 3.5.3 Grounding of a concept).

To find these grounding resources proved to be a problem separated from the technical questions of the thesis. This is a typical question in the field of Information Management – defining a system to organize the personal information and then using and optimizing it. PIM consists of two parts: the technical tool and a concept to use the tool. To use the gnowsis as a tool for personal information management, the author decided to continue existing schemata that were already in use. The existing system is based on folders that are used as classes. The most important class is a project; it is restricted to a certain time and has some resources connected to it.

5.3.1 Projects

A good example of a project is the attendance at a conference. The author decided to create a file folder to represent such a project. A project folder is labelled with the date and name of the project, this is usually the start date or for a conference, the conference date. The label of a folder may be “2003-07-02-IKnowConference”. It is stored in a folder together with all other projects. Browsing through the list of projects, the date and title give a good indication of the content.

In this example, the folder representing the conference attendance is used for many resources. During the preparation, attention is given to the travel tickets and the accommodation. Also a paper may have to be prepared and the timetable has to be checked to decide which talks to attend. The paper is stored in the file folder itself. The address of the hotel is entered in the Microsoft Outlook address book and then linked to the project folder. The interesting talks are entered in a plain text file and stored in the folder. At the end of this preparation, all needed resources are either stored in the folder (and therefore a link appears because of the file system adapter) or related to it through gnowsis links.

When arriving at the location, all resources are linked to the project folder. The address of the accommodation and conference location may be needed, they are linked to the folder and can be found in the browser. During the conference, the timetable and selected talks are interesting; they are stored as files. New notes can be created and new contact addresses may be added to the address book, these contacts can be related to the project using the predicate “acquainted at”. After the conference, emails are written to these new acquaintances. The email addresses may be selected with a RDF query that is run on the gnowsis server.

The use of a file folder to classify resources is just an example; every user may decide to create a custom system to organize resources. The gnowsis proved to be useful in this preparation of a conference.

6 Conclusions

In this thesis we presented the gnowsis, a framework and an application to create a Semantic Desktop. It is a new approach to Personal Information Management (PIM), creating a Semantic Web layer on a personal computer and building applications on top of this layer. To achieve this goal we reused the idea of the web to identify objects via URIs. Since the "Memex" idea of Vannevar Bush in 1945, there have been many approaches to build such a system, the gnowsis is another piece in the tradition of other related projects (see chapter 2).

Our prototype gnowsis implementation was successfully used by the developers for travel preparations and meeting organisation in a number of projects. Emails, appointments, contacts folders and files where connected to each other via RDF triples. A personal web of information was created. This web was be explored using the GnoGno RDF browser, which was developed in this work. It was possible to use a resource on the computer to represent a concept in the mind of the user. The resource and links were used to rebuild the relations of concepts inside the mind of the user with a web of resources inside the computer. A concept that is connected to several resources in the thoughts of the user can also be connected to these resources in the system and these connections can be used to browse through information in a way that is very similar to the associative thinking. A custom vocabulary was used describing facts like "a person attends at a meeting". This vocabulary was integrated into the system, creating a custom taxonomy that can include any resource.

For developers, the application integration features of the gnowsis are interesting: A separate part of the gnowsis is the adapter framework, that was used to extract data from Microsoft Outlook and MP3 files. The extracted data was represented in the RDF notation and was used for further processing. Using the adapter framework, developers are able to build custom adapters to integrate more applications into the gnowsis. Also, the adapter framework can be used separated from the remaining system to extract data from applications.

A user interface paradigm was created by putting the information management functionality in a central server. The GnoGno user interface server was used for information management for existing applications. Integration efforts were minimal because of the fact that only two methods have to be called and these are available for use by an ActiveX object. To prove this ability, the GnoGno was integrated into the Windows File Explorer, Microsoft Outlook and Microsoft Word. These serve as reference implementations for further uses of the system. The information management user interface was used to link resources from different applications. From any resource it is possible to browse to linked resources using the Browser application – independent from the way the link was created.

For idea management and the classification of resources, an existing wiki / weblog system was enhanced with Semantic Web features, integrated into the gnowsis architecture. Links can be created that point from a wiki webpage to a Microsoft Outlook contact and these links are bidirectional, so starting from the contact it is possible to browse to the webpage and pressing the link on the wiki webpage opens the contact using Microsoft Outlook. By creating a special wiki syntax, we were able to enter RDF triples embedded in normal wiki text, using wiki words and links. This enables a new kind of semantic wiki. The weblog feature of the system was used to note conventional diary entries and was also a place to store quick ideas and keep to-do notes.

Through the adapter framework, information entered at different subsystems was integrated into one information space. Queries were executed on this data to extract e.g. the bitrates of MP3 files or the email addresses of persons attending a meeting. The extraction of this data from different applications was done automatically and on demand. The system decides dynamically, which application to contact, based on the URI of the requested resource.

The prototype is released under an open source license and can be obtained at the website of the project, `http://www.gnowsis.com`. It can be used by other developers as an example of how to utilise up to date software to build a Semantic Web application. The source can be reused and modified and the ideas of the system are free to be used in future research and commercial applications.

6.1 Future Work

After publication of this work, the author engages in the process of standardizing the existing system by seeking a team and working together with other researchers and commercial companies to merge existing approaches into a general framework for a Semantic Desktop. The success of this process will depend on the popularity of the gnowsis among Semantic Web developers.

A more frequent use of the system would be necessary and also the development has to continue, so either a distributed open source project needs to be organised or a sponsor for future development has to be found. Especially adapters to extract RDF data from other popular applications are needed, applications such as OpenOffice, Mozilla-Mail and iCalendar.

The user context features applied in gnowsis are only of basic nature. Better algorithms are already available [Schwarz2003] to determine what resources are important to the user, much more user actions can be observed and applications can use this information in different ways. In order to be able to include these algorithms into the Semantic Desktop, the message formats and protocols have to be standardized. A user interface that wants to report what the user does needs a protocol and message format and a receiver where to address the user actions.

Just recently, Tim Berners-Lee in his talk at the ISWC 2003 [Timbl2003], strongly suggested not to invent new URI schemes. In order to comply to this, the suggested RDFP scheme to identify Outlook entries has to be changed to another syntax, preferably http. This has to be done for future releases.

A system like the gnowsis is a good basis for a mobile Personal Digital Assistant (PDA). The gnowsis system would be most efficient, if it is available to the user all the time and all information of the user is stored in it. The system can provide the user interface through a html website, so a mobile version could be realised as a simple storage and processing blackbox device with a mediocre interface. Connected through a Personal Area Network (PAN) other mobile gadgets (like a cell phone) can be used to “browse” the personal semantic web on this personal server.

Appendix

Appendix A: Bounded Description

This is the *bounded description* of a file, the lecture tutorial file *xsl.pdf*. Note the inclusion of the ontology information, the labels and classes of all used predicates are included here. The following text is in RDF/XML syntax and was extracted from the gnowsis using the command line tool and then formatted for better display on this page.

```
<rdf:RDF
    xmlns:j.0="http://xmlns.com/foaf/0.1/"
    xmlns:j.1="http://www.gnowsis.org/ont/filesys/0.1#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >

 <rdf:Description
    rdf:about="file://leo.gnowsis.com/data/uni/xml/skript/xsl.pdf">
    <j.1:bytesize rdf:datatype="http://www.w3.org/2001/XMLSchema#long">
      3322148</j.1:bytesize>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Document"/>
    <rdf:type rdf:resource="http://www.gnowsis.org/ont/filesys/0.1#File"/>
    <j.1:extension>.pdf</j.1:extension>
    <rdfs:label>xsl.pdf</rdfs:label>
    <j.1:filename>xsl.pdf</j.1:filename>
  </rdf:Description>
  <rdfs:Class rdf:about="http://xmlns.com/foaf/0.1/Document">
    <rdfs:label>Document</rdfs:label>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://www.gnowsis.org/ont/filesys/0.1#File">
    <rdfs:label xml:lang="en">File</rdfs:label>
  </rdfs:Class>
  <rdf:Property rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">
    <rdfs:label>type</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:about="http://www.gnowsis.org/ont/filesys/0.1#contains">
    <rdfs:label xml:lang="en">contains</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:about="http://www.gnowsis.org/ont/filesys/0.1#bytesize">
    <rdfs:label xml:lang="en">bytesize</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:about="http://www.gnowsis.org/ont/filesys/0.1#extension">
    <rdfs:label xml:lang="en">extension</rdfs:label>
  </rdf:Property>
  <rdf:Description rdf:about="file://leo.gnowsis.com/data/uni/xml/skript/">
    <j.1:contains
      rdf:resource="file://leo.gnowsis.com/data/uni/xml/skript/xsl.pdf"/>
    <rdf:type rdf:resource="http://www.gnowsis.org/ont/filesys/0.1#Folder"/>
    <rdfs:label>skript</rdfs:label>
  </rdf:Description>
  <rdf:Property rdf:about="http://www.w3.org/2000/01/rdf-schema#label">
    <rdfs:label>label</rdfs:label>
  </rdf:Property>
  <rdf:Property rdf:about="http://www.gnowsis.org/ont/filesys/0.1#filename">
    <rdfs:label xml:lang="en">filename</rdfs:label>
  </rdf:Property>
</rdf:RDF>
```

Appendix B: Mapping RDFS Scheme

This is the RDFS Scheme that describes the mappings between resources and their RDF representation in the AdapterFramework.

```
<?xml version="1.0" ?>
<!DOCTYPE rdf:RDF [
       <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
       <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
       <!ENTITY am 'http://www.gnowsis.org/ont/adaptermapping/0.1#'>
 ]>
<rdf:RDF
  xmlns:am="&am;"
  xmlns:rdf="&rdf;"
  xmlns:rdfs="&rdfs;"
 >
  <rdfs:Class rdf:about="&am;PropMap"
       rdfs:label="Property Mapping"
       rdfs:comment="The mapping of a property. object and subject
are optional. A PropMap is related to a rdf:Property through
hasPropMap">
  </rdfs:Class>

  <rdf:Property rdf:about='&am;defaultPropMap'
       rdfs:label="default Property Mapping"
       rdfs:comment="a default property map for subclasses of
PropMap, Outlook adapter uses this.">
    <rdfs:domain rdf:resource="&rdfs;Class"/>
    <rdfs:range rdf:resource="&am;PropMap"/>
  </rdf:Property>

  <rdf:Property rdf:about="&am;hasPropMap"
       rdfs:label="has Property Mapping"
       rdfs:comment="Property mapping of this rdf:Property">
    <rdfs:domain rdf:resource="&rdfs;Property"/>
    <rdfs:range rdf:resource="&am;PropMap"/>
  </rdf:Property>

  <rdf:Property rdf:about="&am;useRangeAndDomain"
       rdfs:label="use range and domain"
       rdfs:comment="use the range and domain of the RDFS schema to
get the SubjectClass and ObjectClass. Classes that are not wrapped,
will be ignored. true/false. false if omitted.">
    <rdfs:domain rdf:resource="&am;PropMap"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>

  <rdf:Property rdf:about="&am;hasDomain"
       rdfs:label="has subject class"
       rdfs:comment="subject class of the property mapping">
    <rdfs:domain rdf:resource="&am;PropMap"/>
    <rdfs:range rdf:resource="&rdfs;Class"/>
  </rdf:Property>

  <rdf:Property rdf:about="&am;hasRange"
       rdfs:label="has object class"
       rdfs:comment="object class of the property mapping">
    <rdfs:domain rdf:resource="&am;PropMap"/>
    <rdfs:range rdf:resource="&rdfs;Class"/>
  </rdf:Property>

  <rdf:Property rdf:about="&am;propWrapper"
       rdfs:label="property maps to"
       rdfs:comment="This rdf:Property maps to this java-class that
implements PropertyWrapper">
    <rdfs:domain rdf:resource="&am;PropMap"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>


  <rdfs:Class rdf:about="&am;ResMap"
       rdfs:label="Resource Mapping"
       rdfs:comment="The mapping of a Resource. hasClass required
and mapsToResourceWrapper">
  </rdfs:Class>

  <rdf:Property rdf:about="&am;hasResMap"
       rdfs:label="has Resource Mapping"
       rdfs:comment="The rdfs:Class has this ResMap">
    <rdfs:domain rdf:resource="&rdfs;Class"/>
    <rdfs:range rdf:resource="&am;ResMap"/>
  </rdf:Property>

  <rdf:Property rdf:about="&am;resWrapper"
       rdfs:label="class maps to"
       rdfs:comment="maps to the java-class that implements
ResourceWrapper. Value is the name of the Java class. The name has
to be usable in Classloader operations.">
    <rdfs:domain rdf:resource="&am;ResMap"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>

</rdf:RDF>
```

Appendix C: List of Figures

Literature

[Balzert1996] Helmut Balzert, “Lehrbuch der Software Technik, Software Entwicklung”, Spektrum.

[Barreau1995] Deborah Barreau, Bonnie A Nardi, “Finding and Reminding: File Organization from the Desktop“

[Bek2003] Dave Beckett, Jan Grant, “SWAD-Europe Deliverable 10.2: Mapping Semantic Web Data with RDBMSes”, http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/

[Blog2002] Farhad Manjoo, “Blah, Blah, Blah and Blog”, www.wired.com, 18th February, 2002. http://www.wired.com/news/culture/0,1284,50443,00.html

[Booth2003] David Booth, “Four Uses of a URL: Name, Concept, Web Location and Document Instance”. Jan. 2003 http://www.w3.org/2002/11/dbooth-names/dbooth-names_clean.htm

[Brickley2001] Dan Brickley, “Wordnet for the Web” http://xmlns.com/2001/08/wordnet/

[Bush1945] Vannevar Bush, “As We May Think”, The Atlantic Monthly, 176(1), p101-108, July 1945

[Cayzer2003] Steve Cayzer, “Semantic Blogging and Bibliography Management”, Paper at the blogtalk.net conference, May 2003.

[Clark2003] Kendall Grant Clark, “Identity Crisis”, www.xml.com, September 11, 2003. http://www.xml.com/pub/a/2002/09/11/deviant.html

[DCMI] Dublin Core Metadata Initiative, Metadata Terms http://dublincore.org/documents/dcmi-terms/

[DesignPatterns] Gamma, Helm, Johnson, Vlissides, “Design Patterns”, Addison-Wesley.

[Devlin1991] Keith Devlin, “Logic and Information”, Cambridge University Press

[Dorffner1991] Georg Dorffner, “Konnektionismus. Von neuronalen Netzwerken zu einer “natürlichen” KI”. B.G. Teubner, Stuttgart 1991

[Fertig1996] Scott Fertig, Eric Freeman and David Gelernter. “Finding and Reminding reconsidered”. 1996.

[FOAF] Dan Brickley, Libby Miller, “FOAF Vocabulary Specification”, http://xmlns.com/foaf/0.1/

[ForgetMeNot] Mik Lamming and Mike Flynn, “Forget-me-not. Intimate Computing in Support of Human Memory”. Proceedings of FRIEND21, 1994 International Symposium on Next Generation Human Interface.

[Free96] Eric Freeman and David Gelernter, “Lifestreams: A storage model for personal data”, SIGMOD Record (ACM Special Interest Group on Management of Data), 25, 1, pp 80.

[Gem2002] Jim Gemmell, Gordon Bell, Roger Lueder, Steven Drucker and Curtis Wong, “MyLifeBits: Fulfilling the Memex Vision”. ACM Multimedia 2002, December 1-6, 2002, Juan-les-Pins, France, pp. 235-238

[Gibson1984] William Gibson, “Neuromancer”, Ace Publishing, 1984

[Haikonen2003] Pentti O. Haikonnen, “The Cognitive Approach to Conscious Machines”, Imprint Academic, 2003

[Haystack] David Huynh, David Karger, Dennis Quan, “Haystack: A Platform for Creating, Organizing and Visualizing Information Using RDF”, Semantic Web Workshop, 2002

[Hol2003] Rudolf Holzinger, “Komponenten vs. Monolithen”, IT Fokus 4/2003, http://www.it-verlag.de/nc/itf/art/ART_200304.pdf

[Jena] Jena – A Semantic Web Framework
http://www.hpl.hp.com/semweb/jena2.htm

[Jena1] Chris Dollin answering to Leo Sauermann.
http://groups.yahoo.com/group/jena-dev/message/3037

[Kang2000] Hyunmo Kang, Ben Shneiderman, “Visualization Methods for Personal Photo Collections: Browsing and Searching in the PhotoFinder.”, ICME2000, New York City, August 2000

[Koch2000] Traugott Koch, Anders Ardö, “Automatic classification of full-text HTML documents from one specific subject area”, DESIRE II, 2000
http://www.lub.lu.se/desire/DESIRE36a-WP2.html.

[Korz2003] Paul Korzeniowski, “Standards Emerge to Ease Metadata Integration”, www.eaijournal.com, p52-53, February 2003

[Miller2003] Libby Miller answering to Leo Sauermann about identification using WordNet
http://lists.w3.org/Archives/Public/www-rdf-interest/2003Nov/0185.html

[MP3File] Jens Vonderheide, Adam Russel “MP3 ID3 File Tool”
http://www.vdheide.de/java_mp3.zip
http://www.rabbitfarm.com/id3.html

[Nelson1960] Ted Nelson’s Xanadu project, started in 1960.
http://xanadu.com/

[Nelson1972] Ted Nelson, "As We Will Think," On-line 72 Conference Proceedings, vol. 1, pp. 439-454, 1972

[OWL] Web Ontology Working Group
http://www.w3.org/2001/sw/WebOnt/

[Parm] Parmenides of Elea, quoted by Simplicicus and later Burnet. circa 400 B.C.
http://plato.evansville.edu/public/burnet/ch4a.htm

[Pep2003] Steve Pepper, “Curing the Web’s Identity Crisis”.
http://www.ontopia.net/topicmaps/materials/identitycrisis.html
This website has links to important articles about the URI crisis.

[Protégé] N. F. Noy, M. Sintek, S. Decker, M. Crubezy, R. W. Fergerson, & M. A. Musen. “Creating Semantic Web Contents with Protege-2000”. IEEE Intelligent Systems 16(2), pp. 60-71, 2001.
http://protege.stanford.edu/

[RDF] Resource Description Framework, W3C Semantic Web Activity
http://www.w3.org/RDF/

[RDFPrimer] W3C: RDF Primer
http://www.w3.org/TR/rdf-primer/

[RDFS] W3C: RDF Vocabulary Description Language 1.0: RDF Schema
http://www.w3.org/TR/rdf-schema/

[RDFXML] W3C: RDF/XML Syntax Specification (Revised)
http://www.w3.org/TR/rdf-syntax-grammar/

[RFC2141] R. Moats, RFC 2141 “URN Syntax”, 1997
http://www.ietf.org/rfc/rfc2141.txt

[RFC2396] Tim Berners-Lee, R. Fielding, L. Masinter, RFC 2396 “Uniform Resource Identifiers (URI): Generic Syntax“, 1998
http://www.ietf.org/rfc/rfc2396.txt

[Sauermann2003] Leo Sauermann, Gerald Reif, "A survey of RDF Databases", technical report, TU Vienna, 2003

[Schwarz2003] Sven Schwarz and Thomas R. Roth-Berghofer, “Towards Goal Elicitation by User Observation“, LLWA 2003, Karlsruhe, Oktober 2003

[SemWeb2001] Tim Berners-Lee, James Hendler, Ora Lassila “The Semantic Web”, Scientific American, May 2001.
This is the most cited article about the Semantic Web

[Simon1974] Herbert Simon, "How Big is a Chunk?", Science, 1974

[SnipSnap] Stephan J. Schmidt, Matthias L. Jugel. “SnipSnap”, for Fraunhofer Institute for Computer Architecture and Software Technology, available 2003
http://snipsnap.org/space/start

[Stickler2003] Patrick Stickler, “The Nokia URI Query Agent Model, A Semantic Web Enabler”, 2003
http://sw.nokia.com/uriqa/URIQA.html

[Timbl1998] Tim Berners-Lee, “Notation 3”, 1998
http://www.w3.org/DesignIssues/Notation3.html

[Timbl2000] Tim Berners Lee, “Weaving the Web, The Past, Present and Future of the World Wide Web by its Inventor”, Texere, London, 2000.

[Timbl2002] Tim Berners-Lee, “What do HTTP URIs identify?”, July 2003
http://www.w3.org/DesignIssues/HTTP-URI

[Timbl2003] Tim Berners-Lee, “Where to Direct Our Energy “,2nd International Semantic Web Conference, October 23, 2003
http://www.w3.org/2003/Talks/1023-iswc-tbl/slide16-0.html

[TimblFAQ] Tim Berners-Lee, “Frequently asked questions by the Press”
http://www.w3.org/People/Berners-Lee/FAQ.html

[Vinge1981] Vernor Vinge, “True Names and Other Dangers”, 1981
The original cyberspace novel

[Wiki] Various Authors about Wiki Mechanics, “More About Mechanics”
http://c2.com/cgi/wiki?MoreAboutMechanics