



User Interface Concepts for Semantic Information Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Diman Todorov

Matrikelnummer 0225178

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Univ. Prof. Dr. Wolfgang Klas
Mitwirkung: Mag. Bernhard Schandl

Wien, am December 10, 2008

Unterschrift des Verfassers

Unterschrift des Betreuers

I would like to thank all who supported me while I was writing this thesis. Most of all I would like to thank my parents, Bistra and Valentin Todorovi, for making my choice of education possible and helping me to accomplish my degree. Special thanks go to my advisor, Bernhard Schandl, for providing me with a productive environment for a thesis and giving me advice towards successful scientific work. Since an exhaustive list of all the people who contributed to the completion of my degree is not possible without accidentally omitting important names, I kindly ask everybody who helped and supported me through my academic career to accept my gratitude.

Contents

1	Introduction and Motivation	7
2	Background	9
2.1	Semantic Technologies	9
2.2	Overview of the SemDAV Project	15
2.3	Interaction Design	18
2.4	Evaluation Methods	20
3	Related Work	27
3.1	Boolean Query Visualization	27
3.2	Ontology Visualization	29
3.3	Chronological Visualizations	34
3.4	Semantic Desktop	40
3.5	Other Approaches	49
4	Usability Challenges in Semantic Applications	51
4.1	Navigating Large Data Sets	51
4.2	Designing for Diversity	52
4.3	Visualization and Navigation of Ontologies	52
4.4	Visualizing Queries	53
4.5	Semantics of Resource Names	53
4.6	Introduction of New Vocabulary	55
4.7	Addressing of Challenges in Related Projects	56
5	Application of Interaction Design Guidelines in the Development Process	61
6	User Interface Design	66
6.1	The Example of Bob	66
6.2	Overall Appearance	67
6.3	Navigating Large Data Sets	70
6.4	Visualization and Navigation of Ontologies	72
6.5	Visualizing Queries	75
6.6	Introduction of New Vocabulary	81

7	Software Engineering Aspects	82
7.1	Semplorer Architecture	82
7.2	Concurrent Communication Module Architecture	89
8	Experimental Results	95
9	Conclusion and Future Work	97

Chapter 1

Introduction and Motivation

A large problem in today's information society is the sheer abundance of information. While being in possession of larger amounts of information than the competition is an advantage, it is often quite difficult to retrieve useful information from the large repositories. This difficulty does not only manifest itself in inefficient processes but can also lead to stress, dissatisfaction, and even health problems [19]. Various solutions have been proposed in literature: reducing duplicate data, employing specialists to filter and organize information, or automatizing the information archiving and retrieval process. Reducing duplicates and annotation of data relies on some domain experts who organize the data. Human intervention however is not only slow and error prone, it also does not scale to the amount of data that need processing. Software support on the other hand can process large amounts of data quickly but it cannot process the semantics of the data efficiently.

One idea to help software process semantics is to store meaning of information in a machine processable format. This idea is the foundation of the Semantic Web [10]. In the past few years the organization responsible for standards on the web, the *W3C*, has invested much effort in developing standards and methods for storing and representing semantically enriched information. Standards like RDF¹, OWL², and SPARQL³ provide a sound foundation for building backends of semantic applications.

A semantic revolution, resulting in the omni-presence of semantic enrichment of information we access with computers, will not catch on unless it is accessible not only for domain experts but also for casual users. While there are several projects investigating and implementing semantic systems, few of them invest significant effort in the development of generic user interfaces for interaction with and manipulation of semantic meta data.

The thesis at hand proposes a prototypical implementation of a generic semantic user interface. The prototype introduces interaction models, known from other application domains, for working with semantics. The thesis further identifies a number of challenges the semantic user interface designer is facing. The challenges are: how to navigate large data sets, how to design for a diverse user population, how to navigate ontologies, how to visualize

¹<http://www.w3c.org/RDF/>

²<http://www.w3.org/2007/OWL/>

³<http://www.w3.org/TR/rdf-sparql-query/>

queries, how to model the semantic connection between resources and their human readable names, how to introduce terminology required for working with semantic systems. While some of these challenges are unique for semantic systems, most are valid for all information systems. However, to the knowledge of the author, there is no effort in the domain of semantic user interfaces which recognizes and addresses all challenges in a consistent manner. The prototype proposed in this thesis addresses the visualization and navigation of ontologies, the visualization of queries and the navigation of large data sets.

The goal of the thesis is to provide a rapid prototyping platform for semantic user interfaces and to propose user interface patterns which have not yet been considered for the use in semantic user interfaces.

The thesis is organized as follows: Chapter 2 reviews the background of the methods and technology applied in the thesis, Chapter 3 reviews the literature, Chapter 4 treats user interface design challenges, Chapter 5 covers the design process as applied to the prototype, Chapter 6 proposes solutions to some of the challenges identified in Chapter 4, Chapter 7 contains architectural and implementational details, Chapter 8 presents the results of a preliminary UI evaluation of the prototype, and finally, Chapter 9 concludes the thesis and outlines future directions.

Chapter 2

Background

The thesis at hand is based on the assumption that current search technologies have several shortcomings. Before we delve into the arguments supporting this hypothesis it is instructional to define a measure for the assessment of the goodness of a search engine.

The purpose of this chapter is to introduce the technologies and techniques utilized in this thesis. The chapter covers four topics: The first topic are semantic technologies. This section motivates the introduction of semantics in information retrieval and introduces technological and formal background. The second part covers the SemDAV project which is the context in which this thesis was conceived. The third section introduces the design process which served as an organizational frame for the development of the prototype implementation. The last section covers several usability evaluation techniques which were either considered or applied as a part of the design process.

2.1 Semantic Technologies

Semantic technologies, as treated here, are a tool for information retrieval. Information retrieval is defined in introductory texts [14] as follows:

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

The most difficult part of doing this is deciding which documents satisfy the user's information need. Or in other words, which are the relevant documents? In fact, this question is so crucial, that is a central concept in the measure of the quality of retrieved documents.

There are two established measures of the quality of a search result: its *precision* (P) and its *recall* (R). The precision is the ratio of relevant documents (D_R) in a result to the number of all retrieved documents (D). This proportion answers the question of how many of the retrieved documents are relevant. The recall is the ratio of the relevant documents contained in the result to all the relevant documents in the information corpus. The question being answered here is: how many of all relevant documents were retrieved?

A good search engine has results with high precision and recall. Precision and recall are defined as ratios of the number of retrieved and relevant documents. The recall is defined as:

$$R = \frac{\|\{D_R\} \cap \{D\}\|}{\|\{D_R\}\|} \quad (2.1)$$

Precision is defined as:

$$P = \frac{\|\{D_R\} \cap \{D\}\|}{\|\{D\}\|} \quad (2.2)$$

A third measure of search engine accuracy is the *fallout*. It is less frequently used because being the inverse of the recall it conveys no new information. It is defined as the ratio of the irrelevant documents (D_I) in the result (D) to the irrelevant documents in the whole corpus. It describes how much of the documents in a search result are not relevant to the current information need. A formal definition could be written as:

$$F = \frac{\|\{D_I\} \cap \{D\}\|}{\|\{D_I\}\|} \quad (2.3)$$

Assuming these definitions a search engine with high recall and low precision returns many documents (high recall), few of which are relevant (low precision). Important documents however are often not retrieved at all.

The difficulty of deciding which documents are relevant is one of communication between humans and computers. If humans could express their needs in a formal way, computers would be able to serve as perfect information retrieval tools. On the other hand, if computers could understand natural language, they also would be able to produce perfect search results.

Let us demonstrate this difficulty on an example. In the popular web search engine Google, changing the order of two keywords in a query which consists of only two keywords may yield significantly different results. An example of this volatility is shown in Figure 2.1: two searches were performed: one with the query “semantic desktop” and one with the query “desktop semantic”. The latter query retrieved about 7 times as many documents – this variation is a result of merely changing the order of the search terms. The exact mechanics of query interpretation in popular search engines are often unpublished. This is why the author cannot explain how the two queries are different from the point of view of the search engine.

Current search engines are at their limit if they need to combine data from different sources to return a result. Suppose you are looking for all GSM operators which carry a particular model of a cell phone. Eliciting a meaningful

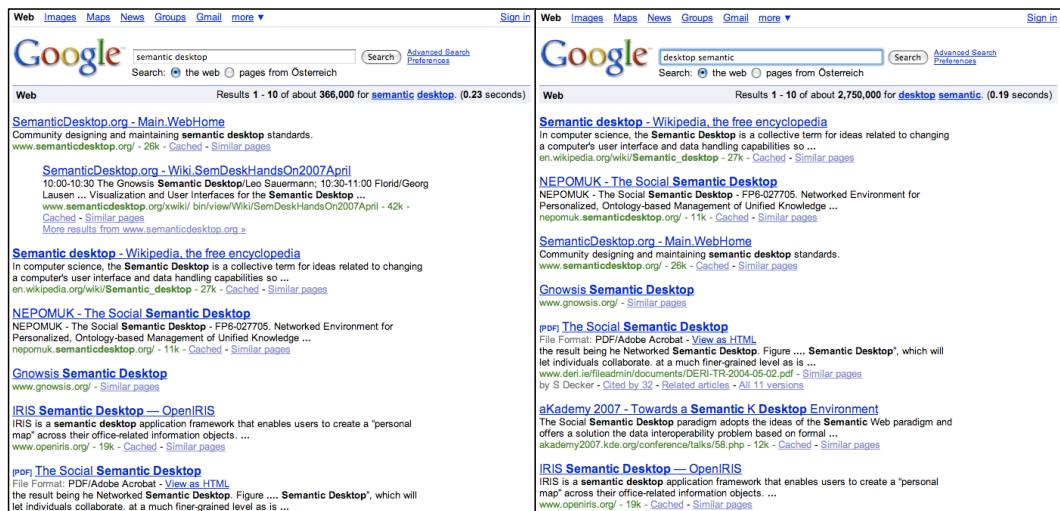


Figure 2.1: Ordering of words in Google.

result to this query from a state of the art web search engine is very unlikely. The needed information is often scattered over several documents. In cases like this there is no other option than retrieving all pieces separately and putting the information together manually. One reason for this deficiency is that the information indexed by web search engines is intended for human consumption, search engines are limited to a purely syntactical analysis of links between documents.

The missing link here is that search engines cannot process the meaning of the information they are indexing and searching. Suppose that the search engine knew of a concept called “GSM provider”. A concept in this sense could be simply a set of instances which fit the description of a “GSM provider”. Suppose further that the search engine knows of another similarly structured concept called “cell phone”. If the search engine also has a notion of a “sells” relation which brings instances of the “GSM provider” concept in relation to the “cell phone” concept, the query asking which GSM providers sell a certain cell phone brand could be answered trivially by analyzing instances of the “sells” relation.

While this supposition sounds nice in theory it has one serious imperfection: the described concepts and relations need to be identified. The lessons learned from artificial intelligence research in the 50’s and 60’s have taught us that computers are not going to be able to process natural language in the near future ([51], p. 22).

What can be achieved however is to represent information in so that computers can easily manipulate, interpret and combine it without actually un-

derstanding it. The problem of lack of understanding can be alleviated to a certain degree by enriching information with semantic annotations. If we define an abstract concept called “GSM provider” and tell the computer that all GSM providers are instances of this concept by enumerating them, it does not matter what this concept is called. The computer does not have to “understand” what a concept means to know what is an instance of it. Following this principle we can model a “cell phone” concept. All we need now is to translate “sells” into something a computer can use as we intend it to. This is solved by defining binary relations between GSM providers and cell phones. A GSM provider x is in relation with a cell phone y only if the provider sells the phone. With these constructs we are able to answer questions about providers selling phones without “understanding” what it means to sell something.

It is instructive to consider what exactly “understanding” means in this context and whether machines can “understand”. In the literature there is no common understanding on whether semantic enrichment enables a machine to understand concepts or to merely manipulate and combine them in useful ways. The question about this distinction was asked and treated both, early and frequently in computer science. For the purposes of this thesis this question is considered irrelevant. Alan Turing, who was one of the first scientists to think about computers understanding natural language, recognized this difficulty. In his seminal paper “Computing machinery and intelligence” [65], in which he introduces the imitation game, later known as the Turing Test, he deliberately avoids addressing the question whether machines can think. He argues that before anybody can claim that computers can or cannot think there needs to be a clear definition of thought and in consequence a definition of understanding. To this day there is no consensus upon such a definition. In an unpublished¹ note Edsger Dijkstra brings the irrelevance of this question to the point with a commonly quoted analogy [16]. About this question he says that it is “a question of which we now know that it is about as relevant as the question of whether Submarines Can Swim.”

Now that we have established, that the best solution is to represent information in a way that can be easily processed by computers, let us describe the involved technologies more concretely. It is important to understand that semantic annotations do not replace the content, they merely describe it.

A lot of the credit for the idea of semantic annotations is attributed to Tim Berners-Lee. In his book “Weaving the Web” [9], he envisioned a web of information in which semantics play a far greater role than in today’s world wide web. The W3C organization, which oversees the development of the web and which Berners-Lee is a director of, has invested significant effort in

¹<http://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD898.html>

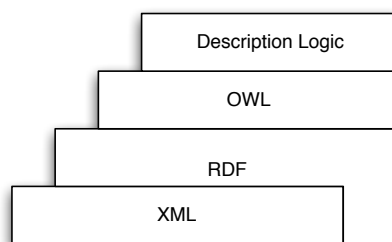


Figure 2.2: A simplified cake diagram of the Semantic Web.

standardizing technologies relevant to the semantic enrichment of the web.

Other deviations from Tim Berners-Lee's original vision of the web which are worth noting are that the web was originally intended to be interactive. The idea was to have a large shared space in which everybody could store and retrieve information. In recent years this idea has proved to be very successful in the form of Wiki's. Another comparably recent development which interprets the original ideas in different ways is the phenomenon of social tagging. This trend towards an interactive and collaborative web without actually changing the underlying technologies is subsumed under the name "*Web 2.0*" [43].

The *semantic* part of the Semantic Web is built in layers (cf. Figure 2.2), from syntax to semantics. Essentially, semantic data is stored in the form of so called *triples*. Triples are statements which consist of a subject, a predicate, and an object. Sometimes statements are described as object, attribute, and value. Technically, both descriptions are accurate. In the above example of GSM providers and cell phones, a triple could be "best provider", "sells", "cheap-phone 321". This method of storing semantic information is quite powerful and flexible. With a few work-arounds even statements about statements can be made. The triple representation of semantics is roughly equivalent to a predicate logic in which only binary predicates are allowed. The down side of the expressive power and flexibility is that predicate logic is known to be undecidable, this means that there are no algorithms for answering questions over predicate logic which are guaranteed to be efficient.

This situation can be improved by trading expressivity for efficiency. When compared with natural language, being able to form triples is roughly equivalent to having a vocabulary. Both, in natural language and in the Semantic Web a vocabulary without rules about forming valid expressions is not very helpful. In natural language these rules are imposed by a grammar, in the Semantic Web the rules are called an ontology. Ontology is a term with roots in philosophy. In computer science however it has a quite different meaning.

A definition of an ontology in the computer science sense is:

An ontology is an explicit and formal specification of a conceptualization [6].

In other words, an ontology contains information about concepts. More specifically, it defines relations between concepts. Such relations can give information about how concepts relate to each other but they can also give explicit information about concepts. For example, suppose that a semantic system has information about a scientific community. Its ontology could then contain the information that the concept “paper” is a subclass of the concept “publication”, which means that every paper is also a publication. It could also contain the information that every “paper” is “authored by” at least one “person”, this is a relation between two concepts. An attribute relation could for example be the “title” of a “paper” – the ontology could express that every paper has exactly one title which is of a textual data type. Other relations an ontology could contain are disjointness and equivalence of classes.

To manage the expressive power versus efficiency trade-off, different restrictions can be imposed on ontologies. If no restrictions are imposed, the case is the same as without an ontology, from a computational point of view – queries cannot be answered with a guaranteed efficiency. We have gained however an explicit and formal specification of the domain of the information. If the restriction is applied, that there cannot be statements about other statements, the expressivity of the framework becomes equivalent to description logic.

Description logic is well researched and there are algorithms which are guaranteed to find an answer to a question in reasonable time. It is a variant of first order predicate calculus. In fact, description logic is a first order logic in which only binary and unary predicates are used. Description logic is the formalization of the OWL language. New knowledge can be inferred from the knowledge base by applying the rules specified in OWL. There are three characteristics which make description logic distinctive. First it is possible to build up a subsumption hierarchy, this can be thought of as a class inheritance hierarchy. Second, it is possible to classify individuals. Questions about the class membership of an individual can be answered within the logic. And third, description logic allows representing relationships beyond simple inheritance. As long as its domain and range can be defined, any relationship between two classes can be defined. Relationship domains, and ranges are inherited from concepts to subconcepts.

The lite version of the ruleset is one which restricts cardinalities of relations to 0 or 1. The main gain of this last version of ontologies is that it is easy to use and implement supporting tools for it.

The standards for representing semantic information are structured in layers. The bottom layer is the Resource Description Framework² (RDF). This layer stores the triples discussed above. A vocabulary which defines semantics ontop of RDF is RDF Schema³. This layer provides constructs for specifying which properties apply to which objects, what values they take, what cardinalities they have and so forth. The next layer is again defined in terms of the ones below it, this layer is the ontology layer. Ontologies are stored in the Web Ontology Language⁴ (OWL). The three kinds of ontology specification languages are OWL Full, OWL DL and OWL Lite. They differ in their expressive power while at they same time they are subsets of each other. This means that every valid OWL Lite document is also a valid OWL DL document, and every valid OWL DL document is a valid OWL Full document.

2.2 Overview of the SemDAV Project

The SemDAV⁵ [55] project is the context in which the thesis at hand was written and in which the Semporer prototype was developed. It was developed at the University of Vienna at the department for Multimedia Information Systems. The running time was from late october 2006 to october 2008. The central goal of the project is to propose a communication protocol and reference implementation for a collaborative *semantic desktop*. The Semporer prototype is a user interface for the SemDAV backend. While its main goal is to implement the SemDAV protocol on the client side and serve as a proof of concept of its feasibility, it is also a convenient platform for the experimentation with concepts for a semantic user interface. To motivate the SemDAV project, let us first outline the meaning of a *semantic desktop*.

Both of the described approaches, adding semantic meta data and the *Web 2.0* have their merits. They share one deficiency though: they rely on the World Wide Web as a foundation. Because of their WWW roots, these technologies are confined in web browsers. Applications running in web browsers however hardly integrate with other applications running on the host machine. Although in recent years web applications have gone a long way, thanks to techniques such as Ajax⁶, they are still a far cry from the responsiveness and flexibility that can be achived with user interfaces which run on the hosting operating system. Desktop applications, on the other hand, are usually single

²<http://www.w3c.org/RDF/>

³<http://www.w3.org/TR/rdf-schema/>

⁴<http://www.w3.org/2007/OWL/>

⁵<http://www.semdav.org>

⁶<http://developer.mozilla.org/en/AJAX>

user centric, and they rarely encourage collaboration as it is seen, for instance, in social tagging. This disparity between the desktop and the web can only be bridged by integrating both, the web browser and the file manager, which is the primary navigation tool on the desktop. Both of these tools can be replaced with a single interface which is both, more generic and more specific. It is more generic, because it does not only navigate either on a file system or on the internet but it brings together both. It is more specific on the other hand, because its main task is navigation and retrieval of data.

It is difficult to make such an interface without making assumptions about user behavior. User behavior however cannot be formally evaluated because a comparable system does not exist yet. Throughout this thesis the behavior of users on the web is used as a guide for design decisions. This benchmark most probably is not accurate. On the other hand, the web was designed with a similar task in mind: to make a large corpus of information accessible, both for reading and modifying, to many users. This, and the fact that web usage patterns are very well researched, motivate the usage of the World Wide Web as a guide for what users will do with a new, more integrated system and how this new system will be utilised.

The available approaches on the World Wide Web were already covered: *Web 2.0* is the more pragmatic solution which however only solves the problem to a certain extent. The Semantic Web is the more expressive and potentially more powerful solution, which has however not yet been widely adopted. Let us now take a closer look at the situation on the other side of the bridge, on the desktop side. The desktop on a computer is a metaphor which is widely accepted and used to allow lay users to interact with computers. Like on a real desktop there are files which are contained in folders, there is a trash bin and so on. The desktop metaphor however was invented at a time in which computers were few and far apart. In today's tightly meshed computer landscape the desktop metaphor is strained to its limits and perhaps even beyond. Nowadays information is accessed not only from desktop computers but also from PDAs, smartphones or other small screen devices. How does a whole desk fit onto the tiny smartphone screen? Information is not uniform, it can be stored in the form of single, self-contained files, emails, URLs or maybe newsfeeds. These forms of information do not integrate with each other on a desktop – the desktop is only aware of files. A desktop in the real world is a single, limited, physical surface, but daily work on a computer is performed in a collaborative process. The boundary between the virtual desktops of different users is blurry and the metaphor does not account for it.

Consider an example: let us say you want to organize the pictures of your last vacation. The pictures were taken in the summer of 2007 in France. In a

tree structure there are two possibilities to store the pictures. The first one is to have all pictures taken in 2007 which has a child node containing all pictures taken in Paris in 2007. The second option is to have a node with pictures taken in France which has a child node containing pictures taken in 2007. In the first scenario it is difficult to list all pictures taken in France. In the second this difficulty is remedied at the cost of introducing another: a list of all pictures taken in 2007 is difficult to obtain.

Web 2.0 applications like Flickr⁷ allow one to organize pictures using tags. The Semantic Web on the other hand is itself a meta data structure which can describe the pictures in high detail while still allowing flexible retrieval. The logical consequence seems to be that the desktop needs this semantic framework. Bringing together the desktop metaphor and semantic technologies is referred to as the *Semantic Desktop* in literature.

The SemDAV project [55–57] aims to provide such a semantic desktop framework. The representation of machine processable semantics is hardly suitable for a user interface. The triples of the Semantic Web need to be wrapped in a layer of abstraction that makes them more accessible. On the web it is widely accepted that every website has its own interface. Interfaces of different websites often have very little in common beyond the underlying technology. On a desktop machine however it is not acceptable to replace the user interface entirely when the user changes their domain. It is traditionally expected that all data on the desktop are navigated in similar manners regardless of their context. To achieve this, an interface is required which is generic enough to be adequate for any domain while at the same time it is simple enough to be used by people with minimal training.

In SemDAV this is achieved by wrapping the triple statement model into a more accessible conceptual scaffold. Desktop concepts and semantic concepts are combined into new concepts which still resemble the originals to a certain degree. Files, for example, can have arbitrary attributes in SemDAV, not only a creation date and owner, but also a language, a MIME type, and so on. These attributable files are called *siles* in SemDAV. Directories are abandoned all together and replaced with *tags* and *categories* as the primary organization tools. Tags behave exactly like tags in *Web 2.0*: users can freely create new tags and attach them to any sile. Categories are organized in so-called *spects*. Spects and their nodes, which are the categories, correspond to ontologies and classes. Siles can also be related to one another with semantic links – *slinks*. The rationale behind the unorthodox naming scheme is elaborated further in Section 4. More information on how users can work with these new concepts can be found in Chapter 6.

⁷<http://www.flickr.com/>

2.3 Interaction Design

Special attention was paid to develop the Semporer interface in a user centered manner. In the literature there is a common understanding that usability is not a step in the software design process but a topic which is present during the whole design cycle. The cornerstones in the design processes described by different authors agree to a large extent – most commonly an iterative prototyping development cycle is recommended. The extension made to the traditional iterative design method is that in the decision making steps user interface aspects are considered explicitly. A coherent and complete method is proposed in Jacob Nielsen’s book “Usability Engineering” [39] – this book served as a guide during the design of the Semporer user interface. Unlike many other human factors experts, Jacob Nielsen is aware of the fact that most projects are developed in resource constrained environments and that formal usability studies after every prototype iteration are a luxury that is usually not within these constraints. He proposes a cost-effective method for designing usable software while pointing out at which steps it is best to invest superfluous resources. His method covers eleven steps (cf. [39], p. 71 ff.):

- Know the user.
- Competitive analysis.
- Setting usability goals.
- Parallel design.
- Participatory design.
- Coordinated design.
- Heuristic analysis.
- Prototyping.
- Empirical testing.
- Iterative design.
- Collect feedback from field use.

These steps are not understood as a strict rule set for the development of software, they serve rather as a rough guide. Some of the points do not apply at all to the development of the Semporer – for example, the Semporer was

never intended for deployment in the field so collecting feedback from users in the field has obviously been omitted. For further details on the design process the interested reader is referred to Nielsen's book [39].

The first step commands to get to know the potential users. It is recommended to visit their site to observe how they do routine work. Special attention needs to be paid on how they currently do the work, how they deal with exceptional cases etc.

Once the designers have gathered a general idea of what the user group will use the software for, Nielsen suggests an analysis of competitive products. He argues that competitive products are in practice equivalent to working prototypes – they can be subjects of a usability evaluation and the flaws can be taken into account in the new product.

Setting clear usability goals is important because different goals may contradict each other. Make the achievement of one task intuitive may make the intuitiveness of another task impossible. To set the goals one needs to know what users intend to do with the application. Potential tradeoffs also have to be identified.

When the requirements of the application have been made explicit in the form of tasks and usability goals, the design phase can begin. The three points referring to the design of the application are not actually steps in a process but recommendations on how to go through these phases more efficiently with respect to usability.

The first proposal is to design several different prototypes in parallel. These prototypes do not have to be fully functional – the main purpose of this step is to experiment with as many different versions of the interface as possible.

Another suggestion Nielsen makes is to involve users in the decision process. Ideally, the involved users would be periodically replaced because when they become too accustomed to the design process, they start thinking in a designer's mindset which obviates the whole point of their presence.

The third suggestion made in the book "Usability Engineering" concerning application design is to keep the design process coordinated. The main goal of a coordinated process is to keep different parts of the application consistent with each other. This is even more important when several people are working on the same components. The coordination should encourage sharing code between components but also provide a clearcut frame for new ideas. It should further support tight collaboration between developers.

Evaluating the application for usability is not considered a one time task performed at the end of the development cycle. Evaluations should be performed frequently so that the fixes of identified flaws also get evaluated. To make frequent evaluations possible a cheap evaluation method is needed. The

evaluation methods considered for the Semporer prototype are described in more detail in the following section.

To produce a mature application, several prototype iterations need to be done. This should always be considered at the onset of a new project. A somewhat martial yet accurate comparison of an iterative development process and a traditional waterfall model is given by Hunt and Thomas in “The Pragmatic Programmer” [26]. They say that the waterfall model, in which one spends a lot of time devising a specification and then develops against this specification, is like heavy artillery. When shooting a large gun a lot of calculations have to be performed prior to the actual shooting and a small miscalculation can mean missing the target by a great distance. The matter becomes even more hopeless if the specification changes during development – hence the target moves. Iterative development on the other hand, where you develop simple prototypes and evaluate them in rapid succession, is compared to firing a machine gun with tracer bullets. The tracer bullets give immediate feedback on the distance between the target and the actual bullets and inaccuracies can be quickly compensated, even when the target is moving.

After a prototype iteration, the prototype is formally evaluated. Nielsen does not give strict recommendations on how this could be done, he stresses however that it is important not to bypass this step. He recommends rating the problems identified with an evaluation method by their severity and addressing them in descending order.

After identifying problems, a new prototype iteration begins. This process is repeated until either an acceptable product is achieved or until the resources run out. An unfinished product which has undergone several improvement iterations is arguably closer to the client’s expectations than one which does not address identified deficiencies of the initial design at all.

The final step is to realize that a project is not completed upon deployment. After deployment feedback needs to be collected from users in the field. It is possible to react to feedback in a maintenance process.

2.4 Evaluation Methods

In his book, “Usability Engineering” [39], Jakob Nielsen presents several inexpensive usability evaluation methods. He recommends however to use more than one of these methods when evaluating an application. The idea is that the cumulative effect of several methods is a reasonable tradeoff between price and performance of usability evaluation. In the following section the evaluation methods considered in this thesis are briefly described.

2.4.1 Heuristic Evaluation

Historically, *heuristic evaluation* has evolved from guideline reviews [36]. Reviewing a user interface with respect to a potentially very large set of guidelines is one of the earliest methods of usability assessment. The main problem of a guideline review is that the number of guidelines easily goes into the thousands. Reviewing a user interface for its conformance to a large set of very specific guidelines is not only a tedious task but, in consequence, also a very expensive process. Molich and Nielsen [36] have distilled 10 heuristics by consolidating guidelines from various sources.

In a heuristic evaluation a user interface is examined and its components are deemed good or bad. The decision which parts of a user interface work and which do not can be made in two ways: either intuitively or based on some set of guidelines. The intuitive approach requires the designer to have wide experience in order to make valid decisions. The guideline method is easier to administer. During the design process a developer will consult a set of guidelines, if the designer follows the guidelines the finished product stands a much better chance to have a good usability. The drawback of the latter approach is that it is very time intensive. Guideline documents are usually several hundred pages long. For example the Microsoft guidelines are published in a book with 594 pages [45], the Apple human interface guidelines are published in a document of 402 pages [7]. Molich and Nielsen [36] have attempted to solve this problem by providing developers with a shorter list of guidelines. They have reviewed several usability guidelines and analysed them for overlapping concepts. The final product are 9 heuristics from which most of the more concrete guidelines can be derived.

The ten guidelines as they appear in “Improving a Human-Computer Dialogue” [36] are:

- **Simple and Natural Dialog** Dialogs should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialog competes with the relevant units of information and diminishes their relative visibility.
- **Speak the User’s Language** The dialog should be expressed clearly in words, phrases and concepts familiar to the user, rather than in system oriented terms.
- **Minimize the User’s Memory Load** The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

- **Consistency** Users should not have to wonder whether different situations or actions mean the same thing.
- **Provide Feedback** The system should always keep users informed about what is going on through appropriate feedback within reasonable time.
- **Provide Clearly Marked Exits** Users often choose system functions by mistake and will need a clearly marked emergency exit to leave the unwanted state without having to go through an extended dialog.
- **Provide Shortcuts** Clever shortcuts – unseen by the novice user – may often speed up interaction for the expert user such that the system caters to both, inexperienced and experienced users.
- **Good Error Messages** They should be expressed in plain language (no codes), precisely indicate the problem and constructively suggest a solution.
- **Prevent Errors** Even better than good error messages is a careful design that prevents a problem from occurring in the first place.

In Nielsen's Book "Usability Engineering" [39] a 10th guideline is presented:

- **Help and Documentation** Even though it is preferable if a system is so easy to use that no further help or documentation is needed to supplement the user interface itself, this goal cannot always be met.

To perform a heuristic evaluation a small group of evaluators examine the user interface and judge its adherence to the guidelines. Nielsen suggests two possible methods to record the results of an evaluation [39]. The first option is to ask every evaluator to produce a protocol of the problems found with a short but detailed outline of every problem. In cases in which a solution is not obvious, the evaluator can suggest one. The second alternative is to have an observer to take notes while the examination is being performed. While this approach appears to be more time intensive because the observer has to be present at every session, it has advantages. One advantage is that the observer will have the results in a homogenous format, and there will be no need to merge notes from different evaluators. Another advantage is that the workload on the evaluators is lighter. If the evaluators are developers it may be desirable to occupy as few of their time as possible to improve usability.

Heuristic evaluation is difficult, studies [40] have shown that even evaluators with experience in the domain of usability research overlook many serious usability issues. It has also been shown however, that different evaluators find

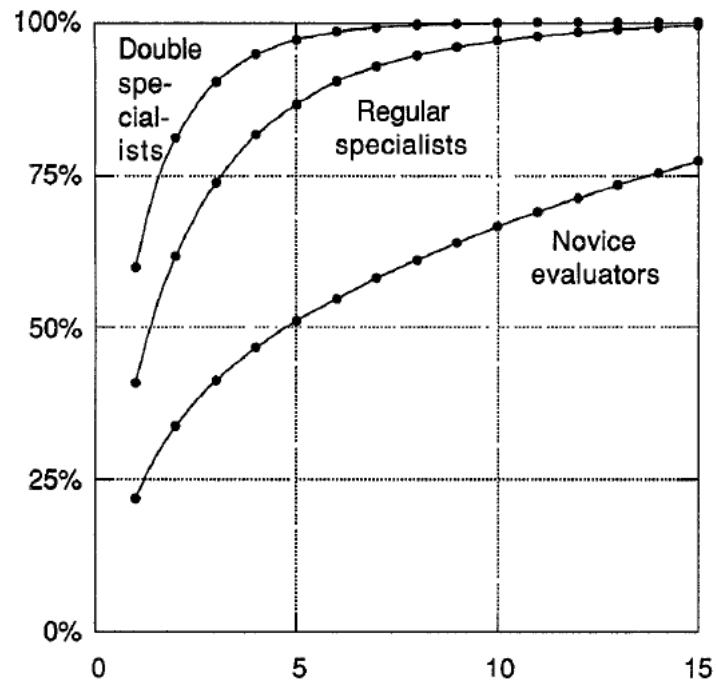


Figure 2.3: Average proportion of usability problems found as a function of number of evaluators in a group performing the heuristic evaluation [37].

different problems. Because of this property of the method superimposing results of different evaluators yields a larger net sum of problems found. It has been shown [37] that depending on their expertise evaluators find between 29% and 61% of all problems. The gain from superimposing results of different evaluators grows logarithmically, Nielsen suggests having a team of 5 specialists for optimal performance (see also figure 2.3).

There are some caveats with the cumulative effect however. In order to keep the overlap of results from different evaluators minimal, the inspections have to be performed independent from each other. If the evaluators were priorly unfamiliar with the method, the tutorial introducing them to heuristic evaluation should be kept in a scope which influences the mindset of the evaluators the least possible.

2.4.2 Thinking Aloud

Originally, the *think aloud method* was a method for psychological research [12]. In such an evaluation users are asked to continuously verbalize their thoughts while they are using the application being evaluated. The result of a think

aloud evaluation is information on what users are doing and why they are doing it. The particular value of this information stems from the fact that it is collected while the user performs the tasks she is verbalizing. Because of its qualitative nature and its relatively informal structure the think aloud method has several pitfalls, one of which is that an evaluator may find it hard to distinguish between information on what the user is doing and rationalization theories of the user. An example described by Nielsen [39] is a user spending more time than expected looking for an input field in a form. When she finally finds the field she might suggest to place the field elsewhere. While users are good at demonstrating usability issues, they are not experts in solving them. For this reason their suggestion for the new location of the field should be taken with a grain of salt.

Another caveat is that an evaluator may have to frequently prompt a user to continue thinking aloud. Thinking aloud is unnatural and may seem strange to users, consequently it may happen that the verbalization stream stops. In such a situation the evaluator has to prompt the user to continue talking. However it requires a certain level of experience to prompt for further information without influencing the evaluation result. If, for example, a user is hesitating while looking at some message, the evaluator might ask her to explain why. If the user has not yet noticed the message, the prompt would influence her attention focus. Other misconceptions and difficulties with this method have been described by Boren and Ramey [12].

Boren and Ramey have identified a wide variation in the way different practitioners administer a thinking aloud evaluation. One of the greatest variations between different experts is what is regarded as “hard data” and what as opinions or explanation attempts of the user. Separating both however is a crucial step in this method. Other differences exist in the interaction with users. Some practitioners ask “neutral” questions, some do not interact at all and others assume a partnership with the user. Often reminders to continue thinking aloud are in the form of questions. For example “John could you tell us why you pressed this button?”. Questions like this interrupt the task the user was working on, they also require introspection. Introspection however is what this method tries to eliminate in the first place. In summary it can be said that the method is treated as a loose guide. Its results are not reproducible and more weight is put on the experience of the practitioner than on a rigorous methodology.

A conduction of this method usually involves two participants, the experimenter and a user. At the beginning of the session the experimenter introduces the user to the experiment. It is helpful to make it clear that the program is being tested, not the user. Often users feel stupid when a task goes wrong, un-

aware that the whole purpose of the evaluation is to identify these situations and improve the application. The experimenter asks the user to perform a given set of tasks using the application. He asks the user to speak his thoughts while she is performing the tasks. It is helpful to conduct a practice round, because talking your thoughts while working may seem unnatural at first. When the user starts completing the tasks, the experimenter will only take notes and observe the progress. The only situation in which the experimenter is allowed to take control is when the user stops talking, in this situation the experimenter would prompt the user to continue talking. Often an audio or a video recording of the experiment is made for later evaluation. Usability experts have reported, that video recordings of struggling users are a great help in convincing developers of usability problems.

2.4.3 Paper Prototyping

A more traditional evaluation method for investigation of user interface design questions is *paper prototyping*. This method is well described in literature (most notably by Snyder [62]), so we will only briefly outline it and compare it to the two other techniques we have discussed. In the most general form of paper prototyping, all possible user interface states are sketched on sheets of paper. The evaluator asks a test subject to manipulate the paper interface and imitates a computer by presenting sequences of user interface states to the test subject. The whole process is logged by a second evaluator and may also be videotaped.

A paper prototype evaluation involves at least three people. Because there is no working prototype of the application, only a paper mockup, one of the experimenters needs to play the role of the computer. Her task is to show different pictures of the user interface depending on what the user did in the last step. A second evaluator asks the user to accomplish different tasks on the pretend application. If the user hesitates at some point or seems confused, the experimenter might inquire why the user is having a problem with the situation. Ideally a third interviewer is protocolling the process. In lieu of a third experimenter, the interviewer can assume the role of the recorder. Like in the thinking aloud method, the evaluation is usually recorded for later reference.

One of the strengths of this technique is its scalability. It can bring benefits even when performed with few or no prior experience. Experienced HCI specialists on the other hand can perform large studies together with a statistical evaluation using paper prototypes. Another strength of this method is that it does not require a running prototype, hence it can support the design process before the user interface is actually implemented. A downside is that it

requires more experience than heuristic evaluation or the think aloud method: since the evaluator is in continuous interaction with the test subject, much more problems can occur than in the other two methods.

One of the advantages of paper prototyping is that it can be applied without extensive theoretical background; of course the quality increases with the evaluator's experience since more elaborate and more detailed scenarios and questions can be used.

Chapter 3

Related Work

The following chapter gives a brief review of literature related to the design of the Semplorer user interface. The covered topics are: visualization of boolean queries, visualisation of ontologies for information retrieval, chronological visualizations, and applications implementing a semantic desktop. Some of the reviewed projects fit into several of these categories. For example, the LifeStreams project is both, a desktop replacement towards a semantic desktop and a chronological visualization method. These border cases were classified by the primary goal of the corresponding project. This is the reason why the LifeStreams project is covered in the semantic desktop section rather than the chronological visualization section.

3.1 Boolean Query Visualization

It is widely recognized in literature [8] that the formulation of textual boolean queries is a difficult task for the average user of an information retrieval system. One of the approaches to alleviate this problem is to construct direct manipulation interfaces for the formulation of boolean queries. In the thesis at hand several approaches found in literature were considered [4, 29, 58]. The two that had the greatest impact on the design of the semplorer are reviewed in greater detail.

The first one is the VQuery language designed by Jones et al. [29]. The V in *VQuery* stands for *Venn* diagrams. The authors of this query language argue that while the majority of users are unfamiliar with boolean algebra, most of them are familiar with the Venn diagram notation of sets. Even users previously unfamiliar with Venn diagrams can be introduced to this notation with little training. The authors have implemented a query interface for a library catalog and have conducted a formal usability evaluation. In this evaluation they have made several findings. First, this type of query interface is only practical for queries consisting of up to three keywords. While this may seem like a limiting constraint at first, it should be inspected from the point of view of user behavior. Two of the most famous studies of search usage patterns [28, 61] have shown that the majority of queries consist of two to three keywords. A second finding is that this type of interface is most effective when users are provided with template diagrams instead of asking them to arrange their own diagrams. In terms of formulation speed Venn

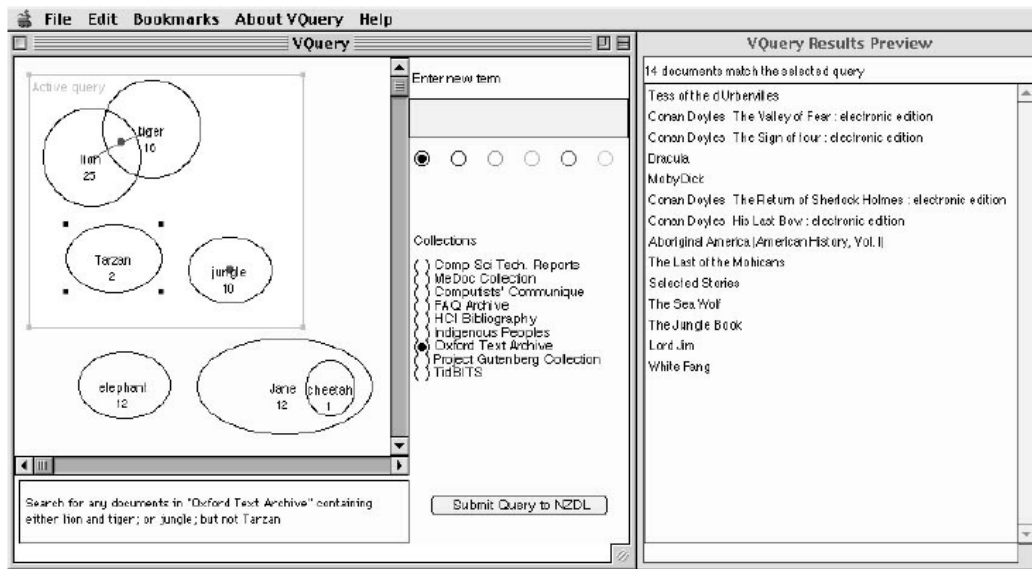


Figure 3.1: Screenshot of the VQuery prototype [29].

diagrams were found to be slower than textual queries with most time spent on arranging the diagram. The third interesting finding is that the qualitative feedback of expert users, familiar with both, textual and visual queries, was that while the Venn diagram interface was slower it was more fun than the textual query interface. A screenshot of the VQuery interface is shown in Figure 3.1.

The second graphical interface which had an influence on the design of the SempLorer was the direct manipulation interface for the AI-STARS information system [4]. This information system was used at the Digital Equipment Corporation for archiving and retrieving user support questions. Previously the information system only had a textual query interface. Queries were formulated in natural language. The system processed the queries, constructed boolean queries and returned some results. The problem with this interface was that the conversion from natural language to boolean queries was largely untransparent to users and yielded seemingly unexpected results. The graphical interface was inserted right after the conversion of the query into a boolean query. Its goal was to provide users with information on how the query was interpreted and give them a chance to modify it.

Figure 3.2 depicts a screenshot of the AI-STARS query interface. The upper window shows the textual query entered by the user. The lower window shows the visualization of the boolean query that was interpreted. In this visualization, the horizontal axis corresponds to a \wedge connective and the vertical

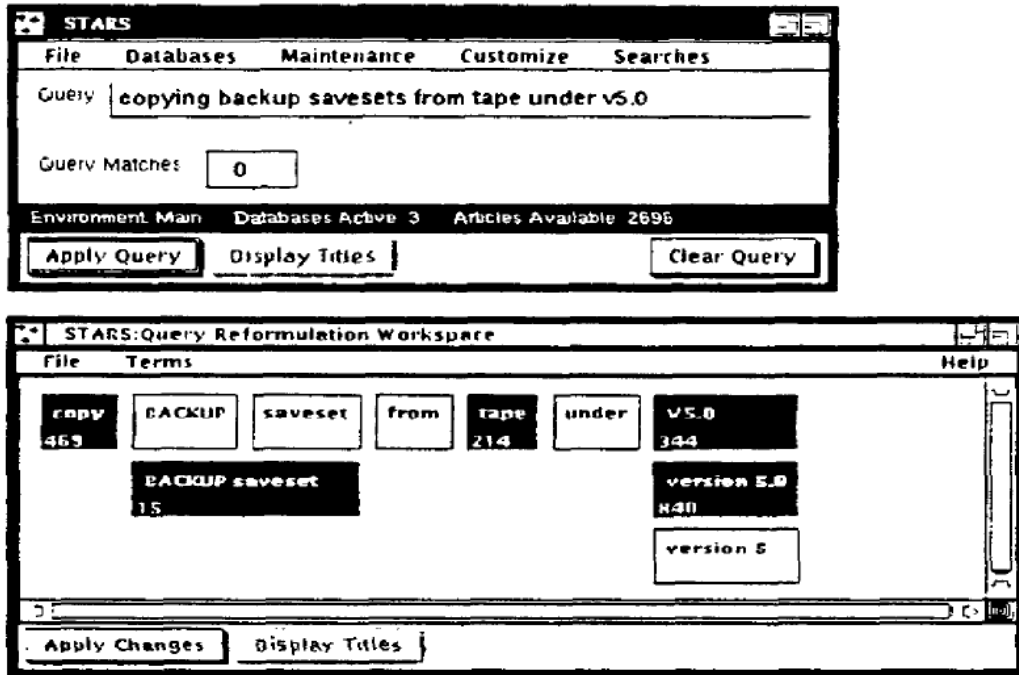


Figure 3.2: Screenshot of AI-STARs visual query interface [4].

axis to a \vee connective. Dark keywords are active and light keywords are inactive. The query is interpreted as if the inactive keywords were not there. The formal interpretation of this visualization is thus:

$$(\text{copy} \wedge \text{BACKUP} \text{ saveset} \wedge \text{tape} \wedge (\vee 5.0 \vee \text{version } 5.0)) \quad (3.1)$$

3.2 Ontology Visualization

While ontology visualization is a broad research field, there are few visualizations suit for interactive navigation and interaction with ontologies for solving information retrieval tasks. In this section we review a survey of such visualization conducted by Katifori et al. [2]. Four implementations are reviewed: SHriMP, TGVizTab, OntoViz and the class viewer in Protégé.

3.2.1 SHriMP

The SHriMP visualization method has its roots in nested tree graphs. The nested graph concept (depicted in Figure 3.3) is extended with various mech-

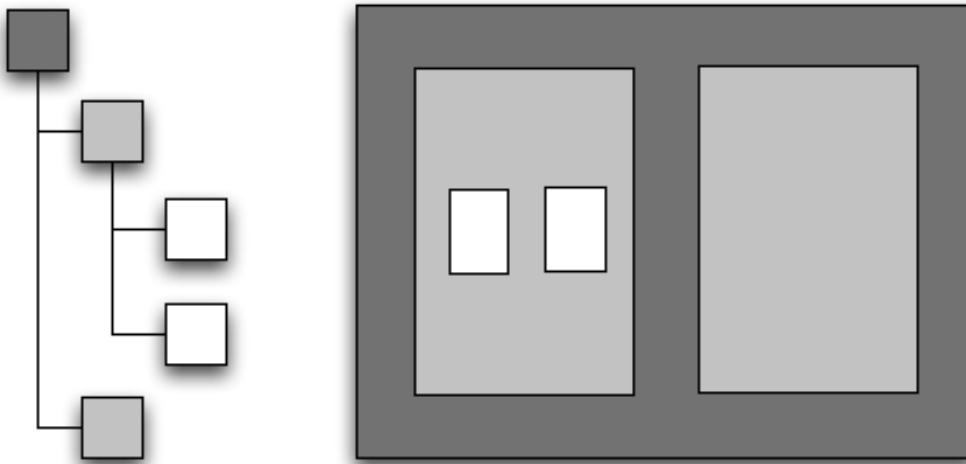


Figure 3.3: A tree and a nested representation of the same graph.

anisms to provide a pleasant usability experience. All manipulations of the graph are animated, this prevents users from getting lost on their navigation paths. Since ontology graphs can become quite dense, great attention has been paid to zooming. The SHriMP interface implements three zooming methods: geometric, fisheye and semantic zooming. The simplest of the three is geometric zooming. This kind of zooming just scales an area on the screen. The second method, a fisheye zoom, also does a scale, however, not a linear one. The closer an object is to the focus of the zoom, the more it will be enlarged, the further away it is, the less it will grow in size. The motivation behind this kind of zoom is that the area of interest grows but the context of the area does not fall out of the viewing area – it is still explicit. The behavior of the last zooming method, the semantic zoom, depends on the kind of object that is being zoomed into. If for example a node represents some text, beyond a certain zoom level this text will be “zoomed” into a text editor so that the user can edit it.

The Jambalaya integration [5] extends SHriMP with various features to accommodate specifically the navigation of ontologies (cf. Figure 3.4.).

3.2.2 TGVizTab

The second interface covered in the survey is TGVizTab [3] (cf. Figure 3.5.). Like Jambalaya it too integrates a foreign technology with Protégé to allow

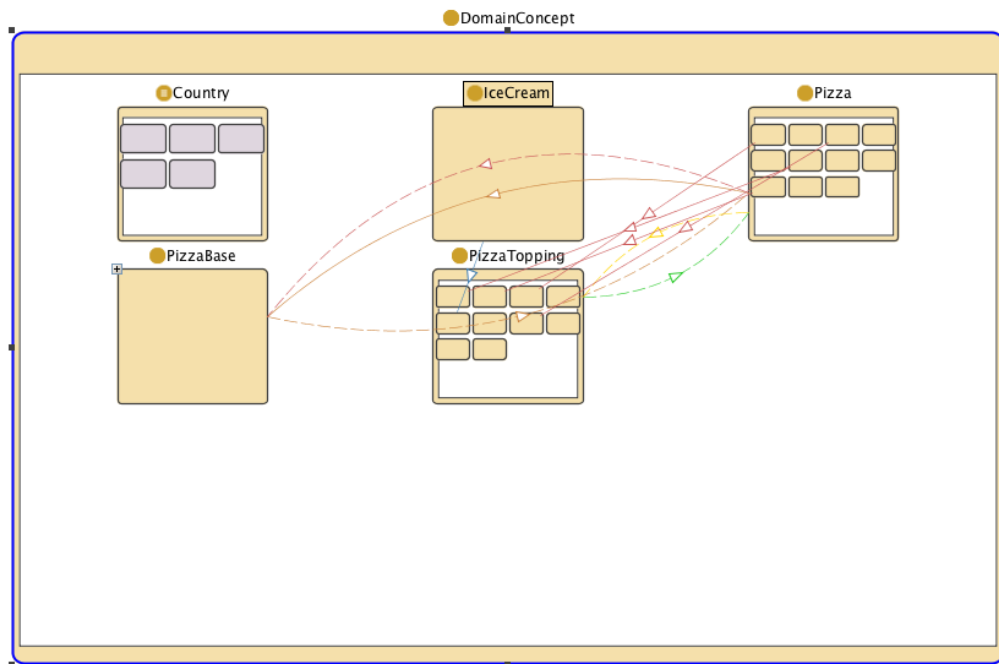


Figure 3.4: Screenshot of Jambalaya in Protégé.

visualizing ontologies. Here the foundation is provided by the TouchGraph¹ library. The goal of TGVizTab is to be a light-weight ontology visualisation. Like the authors of Jambalaya, the TGVizTab creators observe that ontologies are difficult to display as a graph in a usable manner [3]. They address this issue by applying incremental graph browsing and spring layouting. Incremental graph browsing means, that at all times only a subsection of a graph is displayed. For every node on the displayed graph the number of neighbors which are not displayed is shown on the screen. Users can choose to expand the hidden neighbors of a node. Doing this recursively expands only the part of the graph a user is interested in.

As mentioned above, the graph is layed out using a spring model. In this model nodes repel each other and edges attract their endpoints, like springs. The more edges there are between to edges, the closer toghether they are. On the other hand, the more nodes there are in a subgraph, the further away are its nodes. The authors recognize two problems with this mechanism: First, the layout is unpredictable. Just like with real world springs, the graph will seldom find the same equilibrium twice. The second issue is that only connected graphs can be drawn.

¹<http://www.touchgraph.com/index.html>

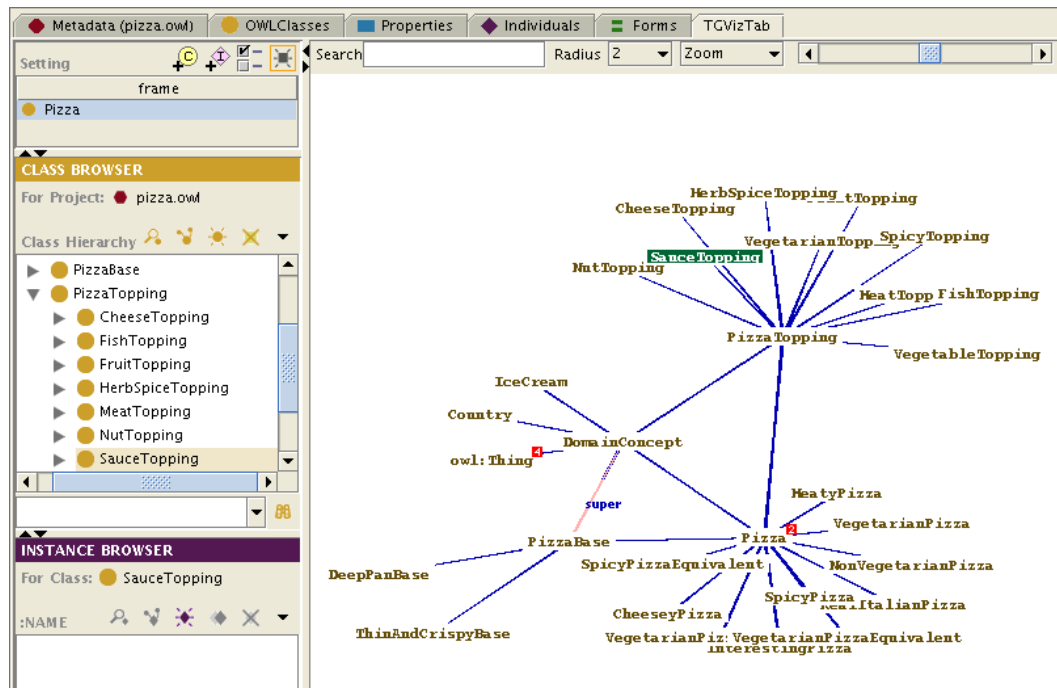


Figure 3.5: Screenshot of TGVizTab in Protégé.

3.2.3 OntoViz

The next approach relies on an external application to perform the non-trivial task of laying out the ontology graph. The OntoViz plugin for Protégé (cf. Figure 3.6.) uses the GraphViz software² to layout graphs. The advantage of using an external application is that the developers can concentrate on implementing interaction concept and outsource the complex task of laying out graphs. The drawback is that the resulting graph is static. It is not possible to drag nodes around, pan or zoom. It is plausible to assume that because of this lack of interactivity this visualization method scored the worst in the comparative usability evaluation of Katifori et al. [2]. There is however a conceptual descendant of OntoViz called OWLViz. While OWLViz also relies on GraphViz for laying out graphs, the level of interactivity is a lot higher. It does not make it possible to drag, pan and zoom, but it does allow to navigate an ontology tree incrementally. Since OWLViz is not covered in either the comparative evaluation or in other literature, it will not be considered any further.

²<http://www.graphviz.org/>

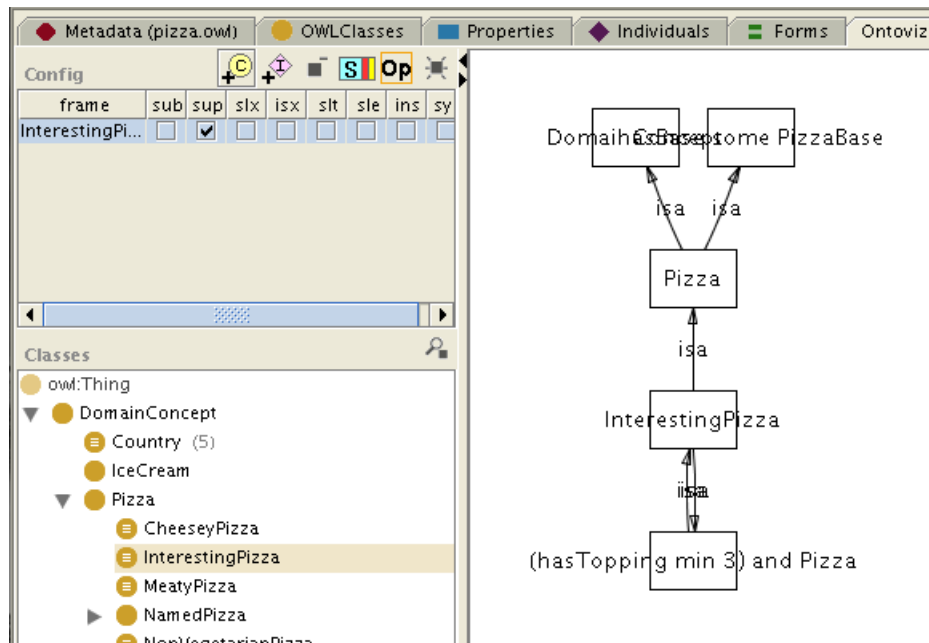


Figure 3.6: Screenshot of OntoViz in Protégé

3.2.4 Tree-Graph Visualization

The last visualization method for ontologies that will be discussed is often not regarded as a visualization method in its own right. This approach is visualizing an ontology as a simple tree widget (cf. Figure 3.7.), like the one found in the Windows file explorer or the Apple Finder. With respect to ontology visualization, this widget has several shortcomings. For one, the tree viewer is limited to visualizing trees, ontologies however are directed acyclic graphs. If one node has more than one parent, the tree widget cannot display it adequately. The second problem is that only inheritance relationships can be visualized, other kinds of relations between ontologies cannot be displayed at all. The first problem is solved by displaying a node with multiple parents as a child of all its parents. The second is simply neglected. Although this method may seem quite naive, it scored best in the comparative evaluation. In spite of its high score, users pointed out several deficiencies. For example: only visible classes can be searched, to expand or retract a node users have to click on the arrow beside the label, there is no button to expand and retract all nodes.



Figure 3.7: Screenshot of the ontology class browser in Protégé.

3.3 Chronological Visualizations

In this section Lifestreams is the central project. It is not only the earliest formal treatment of chronological visualizations in user interfaces known to the author but it is also widely cited in literature. The remaining two projects described in this section pick up the ideas proposed in the Lifestreams project and develop them further.

3.3.1 Lifestreams

Lifestreams [20] is a project originally initiated in the mid 90s. It is one of the more successful approaches to breaking with the desktop as the main metaphor for interaction with a graphical operating system. Adhering to the desktop metaphor may be easier for the many users who are comfortable with it. On the other hand it prevents the development of any genuinely new ideas. The authors argue that files and folders are concepts which were developed before computers became pervasive and massively connected. For this reason they are inherently inadequate tools for organizing the amounts of information users have to cope with on a daily basis. The reason is that a hierarchical folder structure in which files can only reside in only one folder forces users to categorize all their data into distinct categories. Categorization of documents into distinct folders is the hardest information management task users are facing. Some researchers go even further and say that categorization is a flawed psychological process [18]. In Lifestreams they are replaced with a time ordered

stream of documents. The authors claim that while the Lifestreams effort may not have influenced developments in human computer interaction directly, it has been a good predictor of the general direction of HCI developments.

The Lifestreams interaction model revolves around three key ideas. These are: transparent archiving, reminding and summarizing. Old information is less valuable than new information but still often essential. The desktop metaphor provides practically no support for archiving infrequently used information. Users of desktop oriented interfaces have been observed to place documents at strategic locations as a reminder for the following day. This reminding mechanism is unreliable at best. The idea of a summary is that a document or a group of documents are displayed in a window in a summarized form so that users only need to look at the summary to get an idea of the content of the documents. Summarizing minimizes the amount of information that has to be processed to find a particular document. The Lifestreams model attempts to address these issues by avoiding the usage of metaphors. Interaction concepts are based on *virtuality* instead.

One issue with files is that on a metaphorical desktop all files and all folders need to have a name. On a real desktop however informal documents do not have names. When we take a random note we seldom give it a title. Informal electronic documents however need names. Many users end up giving them names like “Untitled” or “draft” which make later retrieval very hard if not impossible. Other problems of the desktop metaphor are rooted in the metaphor following the original too closely. For example, every file needs to reside in a single folder. Although there are mechanisms to circumvent this limitation, they are barely supported in mainstream applications. The save dialog of a text editor will only ask for one storage location of a document. Another restriction is that electronic desktops, like their real counterparts, have a static organization. Since this organization is just one possible representation of the data, rendered for the benefit of a user, it does not need to be static.

The ubiquitousness of the desktop metaphor has more subtle implications also. In recent years there has been a great leap towards mobile computing. Even the simplest of today's cell phones have a manifold of the processing power of the computers for which the desktop metaphor was designed. There is however no adequate translation of the desktop metaphor to mobile displays. What is the unmetaphorical counterpart of a portable desktop?

The remainder of this section will elaborate the Lifestream concepts and interaction models. Figure 3.8 depicts a screenshot of the Lifestreams user interface. A Lifestream is a time ordered stream of documents. The real world analogy best describing a Lifestream is a diary. Each document in a Lifestream is stored at the time it is first created or received. It is however also possible

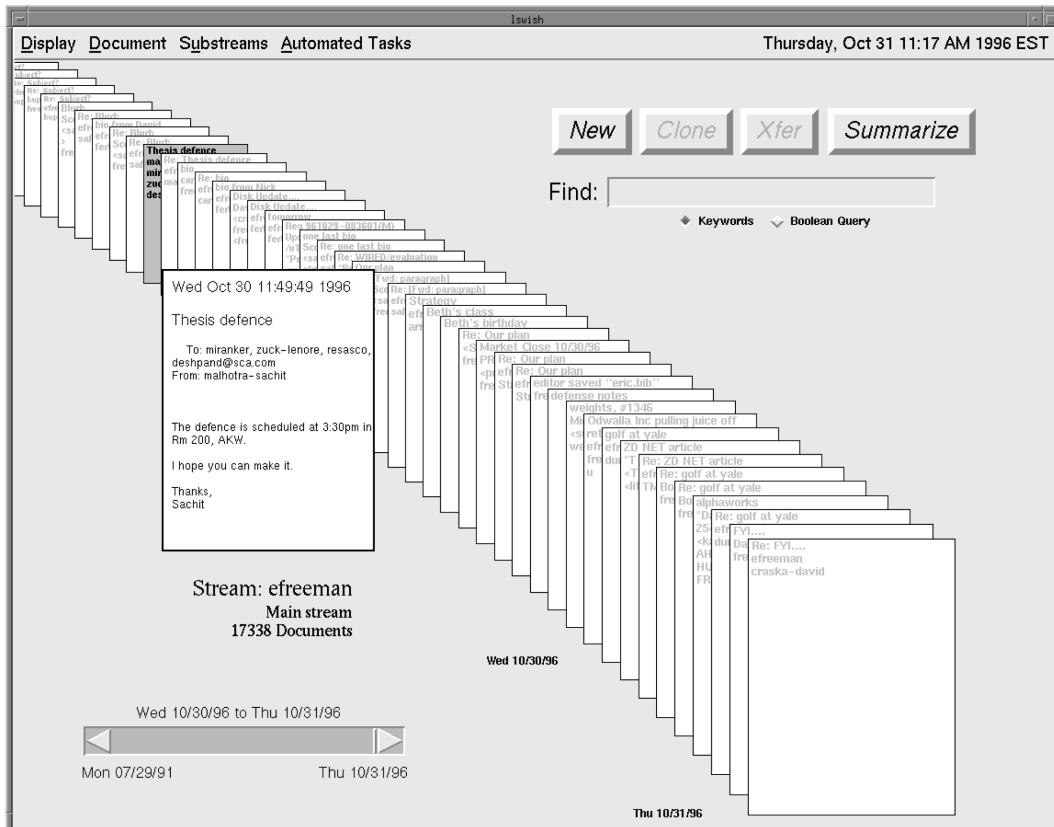


Figure 3.8: The Lifestreams user interface.

to move to the future of a Lifestream and create a document there.

Lifestreams are manipulated using a few well defined operations. The operations for creating new documents are *new* and *copy*. The *new* operation appends new documents at the head of a Lifestream. The latter operation, *copy*, takes an existing document and adds a duplicate to the head of the stream. It is worth noting, that new documents do not have to be named.

Documents in a stream are retrieved using the *find* operation. This operation is operated by entering boolean queries in a text entry box. Every query entered in this box can be persisted to a virtual group. The group is updated when new documents matching its query are added to the stream. After retrieving, documents are modified and viewed with external applications. This process is orthogonal to creating and retrieving documents.

Summary previews of documents is another key feature of Lifestreams. Summaries can be generated at two levels in the Lifestreams environment. The first kind of a summary is a document summary. When a user hovers with the mouse cursor over a document a thumbnail of the document is displayed. It is notable that the manipulation functioned in real time which is remarkable considering when the first prototype of Lifestreams was written. The second level at which documents are summarized are groups. In group summaries one preview representing all document in the group is generated. To sum it up, the authors say that while they have experimented with several summarization methods, many questions remain open.

The original Lifestreams prototype was implemented in a client server architecture. It allowed users to send each other documents over email. The incoming document would be added to the head of the receivers stream. The authors implemented a graphical client, a text console, a pda and web client for the Lifestreams server.

Lifestreams were conceived to assist users in communication, reminding, tracking contacts and many other every day scenarios. In the Lifestreams environment one user can send a document to the future of another user's stream – this would amount to a reminder. Aging documents move out of the users view and beyond a certain threshold they are stored in an archive.

Although some prototypes allowed sorting by criteria other than time, time is the main ordering of Lifestreams. The rationale behind this design decision is to allow using the historical context of a document as the main index in the retrieving process. The authors believe that historical context carries more information than the location of a document in a specific folder and carrying a given name.

The authors collected feedback about the Lifestreams application from three user populations: the staff in their department, readers of popular press,

and potential clients who were shown a demonstration. They also administered a formal evaluation using a structured questionnaire. The response was found to be largely positive. The authors acknowledge however that the largest problem of the Lifestreams application is the lack of integration with other systems.

On a closing note, the authors state, that in the battle against information overload a good browsing engine is at least as important as a good search engine.

3.3.2 TimeScape

The most noteworthy spin-off of the Lifestreams project from a design point of view is the TimeScape [50] application. TimeScape's greatest conceptual innovation in comparison to Lifestreams is that the author introduced context distances in addition to temporal distances. In TimeScape users can place documents which are treating a similar topic close to each other. The idea of spatial vicinity as a measure for context vicinity is derived from Malone's famous study on how people organize their desks [33].

The TimeScape application is intended to replace the native desktop environment. It is used in place of the Finder on a Mac or the Explorer on a Windows machine. Similarly to the LifeStreams project described in the previous section, TimeScape archives all user data making the assumption that disk space is cheap and retrieval overhead is small in most practical cases. The main navigation paradigm in TimeScape is chronological. The users browse their data on the visualization of a timeline. TimeScape provides several visualization techniques for time: for example a timeline or a calendar view. Convenient operations like panning, zooming, double clicking for opening files and others are implemented. Unlike the LifeStreams user interface, TimeScape benefits from today's more powerful graphical programming hardware and libraries. One innovation they made is to visually fade files which are further in the past. Searching is not a central topic in the TimeScape project, the authors provide only a full text keyword search. They do however implement the notion of dynamic filtering: files that do not match the text query are not displayed.

The authors see the timeline interface as a useful paradigm for information management on handheld devices. Small display sizes are limiting in terms of navigation paradigms and a timeline visualization can be implemented to use screen space sparingly while still being intuitive. However, formal usability evaluations have not been conducted so there is no evidence in support of this claim. Figure 3.9 shows a screenshot of the TimeScape user interface.

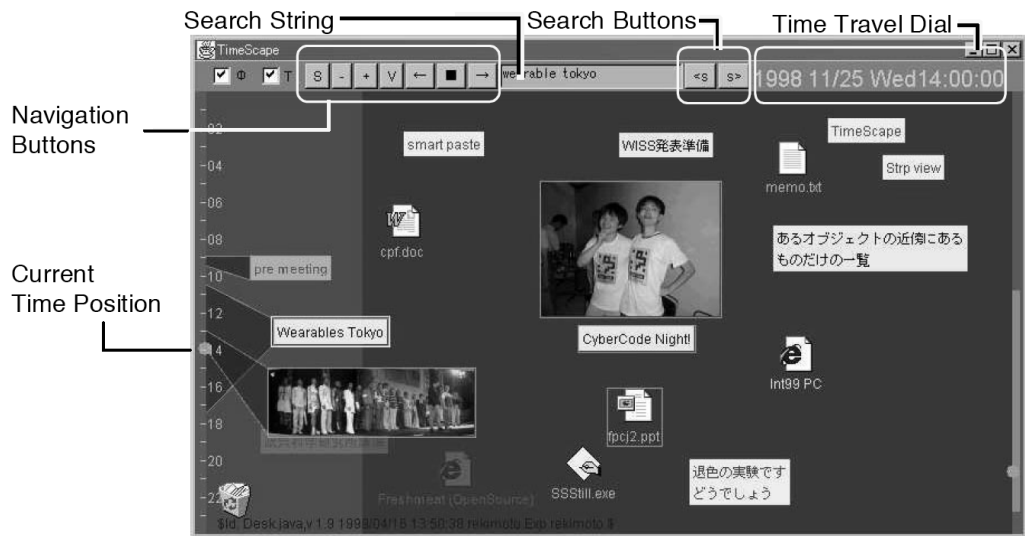


Figure 3.9: TimeScope screenshot [50].

3.3.3 Dynamic Timelines

Another project from a slightly different domain are Kullberg's dynamic timelines [31]. The dynamic timelines are used to visualize a database of historical photography. Kullberg has approached the problems of timelines by visualizing in a three dimensional space. Figure 3.10 depicts a screenshot of the three dimensional interface.

Kullberg's dynamic timelines are not directly related to the other applications in this section. The timelines have a very specific application domain and a static database. In spite of this difference, Kullberg documents many challenges also present in the Lifestreams, Timescape and Sexplorer applications. The probably greatest challenge for timelines is the visualization of dense clusters. On a timeline there tend to be periods in which many data are stored and periods in which there are no data at all. A linear spacing of markings on the timeline thus means that if the interface is zoomed to display a dense cluster, the user would have to pan for a long time to reach another cluster. On a zoom level at which distinct clusters can be made out, the instances in the clusters are usually too cluttered to interpret. This situation makes for frequent zooming operations, which is the second challenge similar in all timelines. When zooming unevenly spaced data it is easy to *get lost* in the data, so to say. Or in other words, when zooming into a dense cluster it is easy to lose the context of the area which has been zoomed into.

There are however also conceptual differences between the visualization of

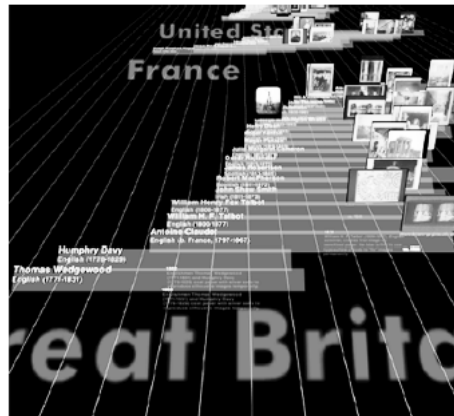


Figure 3.10: Dynamic Timelines fourth prototype screenshot [31].

historical information and the information on a desktop. In history, objectivity plays a large role. It is important that historical events are not emphasized beyond proportion in the visualization. On the desktop on the other hand, a certain degree of subjectiveness is very welcome. For example, data that are not related to the current project do not need to take up the same amount of screen space as data which the user is currently manipulating. On the desktop, a common heuristic for the relevance of data is the date of the last modification. In a historical context it would be unthinkable to lower the relevance of events depending on how long ago they happened.

3.4 Semantic Desktop

There are several projects attempting to achieve goals comparable to the ones of SemDAV. These goals can be subsumed under the common vision of a Semantic Desktop (cf. Chapter 2). To remain inside the scope of discussion only projects beyond a certain maturity level will be treated in detail.

3.4.1 Haystack

Haystack is a project at the MIT [27]. It was started in 1997 to research methods of coping with information overload. The goal was to develop a flexible framework which integrates information from different sources in a consistent manner. Originally Haystack was developed on top of a custom data model but later it was migrated to an RDF store. The main innovation of Haystack from a user perspective is its flexible user interface which can be customized to every user's needs. The configuration of the user interface

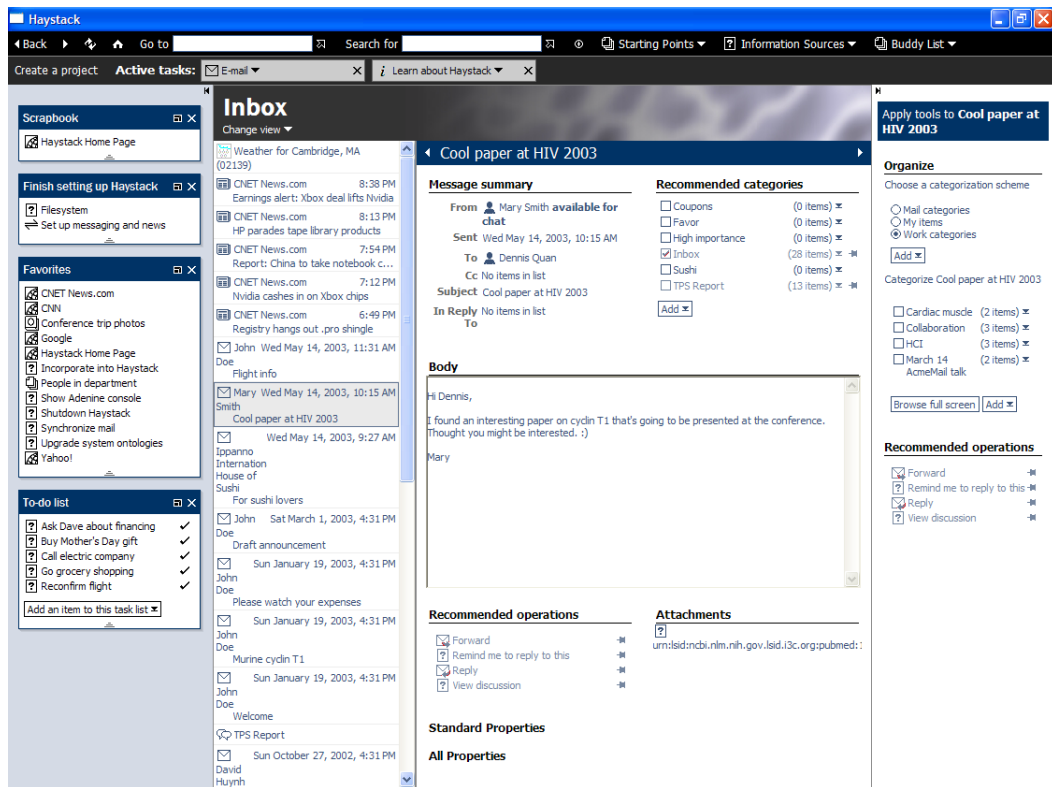


Figure 3.11: Haystack user interface [24].

is stored as an RDF graph using the same facilities which store and retrieve the meta data managed by the Haystack application. Haystack is designed to work with and within the Semantic Web, its main focus lies on aggregating data from different information sources and integrating them in a personal semantic repository.

In order to keep the manipulation of Haystack's user interface simple several concepts are applied [47]. The probably most important one is the notion of *collections*. Collections in Haystack play a similar role as folders in a file system. Collections however are more generic than folders. They are aggregations of objects, but this is as far as the similarity goes. Collections are not strictly hierarchical and objects can be in several collections at the same time. With symbolic links on Unix and shortcuts on Windows, folders expose a feature which provides a similar functionality. The user interface support for links and shortcuts however is very rudimentary. The authors of Haystack expect that the greatest bulk of objects classified into several collections will be automatically performed by agents. An agent can represent a result of a query as a collection for example.

The Haystack user interface is recursively constructed from *parts*. A part can consist of more parts and can render RDF meta data. The structure of user interface parts is itself defined in RDF. Different part types persist their settings using their own, custom ontologies. Using generic view parts to visualize RDF data makes for a very consistent user interface. For example, all user interface elements are designed as sequential lists. Another example is that all elements of such lists can be dragged and dropped onto other elements, which may be lists. The main advantage of this design is that once a user has learned to work with one component, she can reuse her skills when interacting with another, possibly unknown component.

On a higher level of abstraction the user interface is not described in RDF but in a scripting language which is translated to RDF. Using a scripting language makes the development of the user interface more comfortable as it automates the usage of repetitive patterns. The scripting language has been developed specifically for the Haystack project, it is called Adenine [46]. The Haystack authors describe Adenine as a Lisp dialect which does not operate on lists of pairs but on RDF triples. Because the language is implemented in Java it allows tight integration with modules written in Java. Another similarity between Lisp and Adenine are *continuations*. Continuations are non preemptive operations which can be persisted before they have completed. Continuations provide the foundation of user interface continuations as described by Quan et al. [49]. These are used similarly to macros. If for example a shopping basket needs a user to enter their identification before they proceed shopping, a continuation can be created after the user enters their identification. In subsequent shopping sessions the user can start from the persisted continuation eliminating the need to reenter their identification. Using an interpreted language for the construction of the interface allows to make changes to the user interface at runtime. This in turn enables user interface designers to concentrate on the design process itself neglecting technicalities. To illustrate the relationship between user interface design and programming the Haystack authors paraphrase designers as interior designers and programmers as carpenters [27]. The user interface construction process however still bears enough marks of semantic technology to encourage designers to think in a semantical mindset when developing the user interface.

Haystack intends to be the single point at which all personally relevant information streams are merged and integrated. It brings together email and RSS but also calendars and potentially many other sources. Agents, which are small pieces of software performing information manipulation and retrieval tasks, can extract logical structures by reasoning over the aggregated information space. Since Haystack does not really delete meta data but instead marks

it as deleted, agents can also reason over information that has been deleted. They can even reason over information that failed to be written. Haystack agents are written in Adenine.

The authors also acknowledge some problems in the Haystack project. One of them is finding a suitable RDF database. Currently the authors are using a database written specifically for Haystack. It is written in C++ and optimized for Haystack's needs [27]. The authors however acknowledge that the database is only a temporary solution and are looking for better alternatives. Another unsolved issue is the problem of access control and privacy. While there is ongoing work at W3C attempting to devise standards for trust in the Semantic Web, there are no implementations which can be used in a project. Haystack classifies this issue as out of scope and circumvents it [27]. The last problem Haystack is facing is that while the user interface has great potential it has not been designed by a usability expert [30].

3.4.2 mSpace

The mSpace project [59] is being developed at the School of Electronics and Computer Science of the University of Southampton, U.K. Its goal is to make browsing information easier. The authors provide a catchy description on their website³ which sums up idea behind mSpace: "Imagine Google on iTunes". The project is tackling the information retrieval problem with facilities for explorative navigation [59]. Currently the mSpace framework is only suitable for browsing preprocessed domain specific data. Its long term goal however is to become a generic Semantic Web browser.

When searching information users hardly ever get their query right at their first attempt. The search for information is usually one of iterative refining. Finding a query which to refine is easier when the user has background knowledge on the information they are looking for. Using search engines like Google without prior knowledge may be a frustrating experience. The mSpace team sums up this scenario with fictive user's statement: "I don't know anything about classical music, but I know what I like when I hear it". This fictive user will want to explore the information space. The user will initially not be able to formulate a query which will return music they like. Along their navigation path they will listen to music in order to decide whether they are currently in a musical category which they like. By listening to representative music from different musical categories the user will start to create associations. The crucial point here is the assumption that the human brain models an information space by association. Allowing associative navigation is thus

³<http://mspace.fm/whatis/>

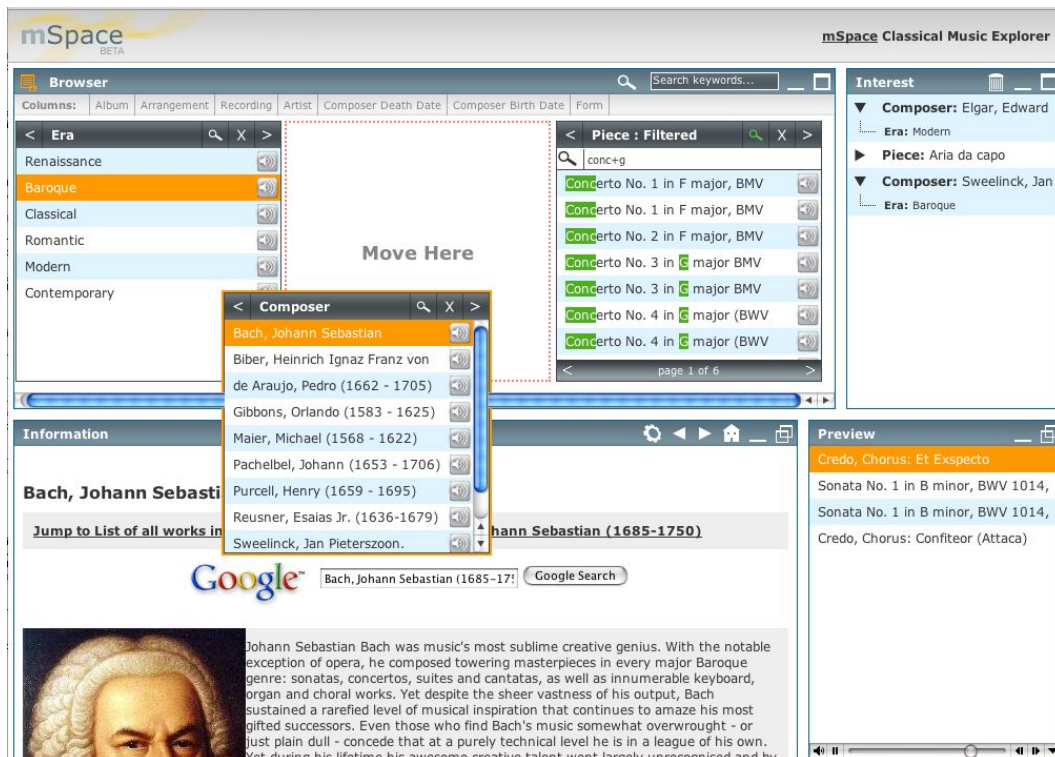


Figure 3.12: mSpace user interface [41].

much closer to the users mental model of the information than a plain index. mSpace supports this process by supporting access to material at any point on an exploration path, by emphasizing relations in the information, and by providing multiple starting points for a possible navigation path.

The central part of the mSpace interface is a multi column view (Figure 3.12). Each column displays one dimension of the data. Multicolumn views as a representation method for high-dimensional data are not a novelty. The innovation here is that the columns can be freely manipulated. Users can add new columns, remove columns, and swap the location of columns which are currently displayed. Highlighting an entry in the leftmost column expands all data which lie at the highlighted point of the dimension represented by the leftmost column. Referring to the classical music scenario, if the leftmost column is displaying musical periods then selecting one musical period will fill all columns to the right with data taken from this period. If the second column displays composers then selecting “baroque” in the first column will fill the second column with composers who have composed pieces classified as baroque. The mSpace team has experimented with various refinements of this basic concept.

One crucial refinement is the introduction of *preview cues*. These cues are the mechanism supporting the formation of associations. The purpose of a preview is to give the user a general idea of a category without having them browse the category. This is achieved by selecting an instance of the category which has been found to be representative. Naturally this approach works best with multimedia content. In the classical music scenario a preview cue for baroque might be a Vivaldi concert. Preview cues have their foundation in the concept of information triage [34]. To make the mechanism work it has to be accessible. The authors have chosen to trigger it with simple mouse gestures. Brushing with the cursor over a label initiates the playback of the preview. The largest obstacle in implementing preview cues is making the choice of representative instances. The authors experimented with cues chosen by experts and with cues which were randomly chosen from the category. They have investigated algorithmic solutions but found no satisfying option [66].

Bringing the idea of preview cues further, mSpace implements the notion of *info views*. An info view is a pane which displays visual and textual details on the currently selected category. The authors believe that providing peripheral information further assists the building of associations.

Another thing that was also implemented is the use of *numerical volume cues*. A numerical cue would display the number of composers falling into every period by appending the number to the textual representation of the period in the column displaying periods. If for example there are 328 composers in the database who have written baroque pieces then the text in the period column would be “baroque (328)”. Of course this interpretation is not as straightforward as it seems. If there is a third column which lists all pieces then the number 328 might mean composers but it also might mean pieces. A rigorous treatment of numerical volume cues has been published by the authors [66].

3.4.3 Nepomuk

The Nepomuk project [25] is a large project with the aim to create a standard and a reference implementation for a social semantic desktop. In the current desktop environment every user is on their own island. In the Nepomuk vision on the other hand, there is a standardized communication layer which blurs the boundaries between applications and physical locations. Instead of e-mailing a document to a colleague it is made available through the Nepomuk framework.

The larger Nepomuk project is a follow-up project to the Gnowsis [52]. Gnowsis is a prototype implementation of a semantic desktop framework. The Gnowsis can be described in short as an operating system wide classification system. It can be seen as an alternative or an enhancement of the file and folder

organization. The classification is realized by defining a Uniform Resource Identifier (URI) for every resource on the desktop. Links between these URIs can then be created and stored in a central server. By prolonged application of this process a personal Semantic Web is weaved. Because the resources for the Gnowsis project were very limited it was implemented as a single user system.

The Gnowsis is a thin layer on top of a traditional file system. It provides a uniform resource identification (URI) for any resources a user might want to retrieve and allows annotating these resources. The Gnowsis does not replace traditional applications, it rather extends them. The meta data in this framework is stored in a central repository. The main reasoning behind this design decision is that it allows treating links between resources as bidirectional. In a decentralized repository system, where meta data is stored along with the resource itself, it would be easy to make a link from a resource *A* to a resource *B*. Finding all resources *B* which have an *A* linking to them however would mean that one would need to look at every single resource and check whether it links to *B*. This mechanism essentially makes it possible to link resources from different applications with each other.

The meta data in the Gnowsis is stored as RDF triples. It is extracted from existing documents, transformed to RDF and buffered in the central Gnowsis repository. The extraction is done on the fly every time when a retrieval operation is requested. This method results in slower response times but yields consistent data. If a user queries the Gnowsis for all music files of a certain genre, the Gnowsis will look at each file it knows about, extract its genre and compare it with the requested genre.

Introducing the Gnowsis into the daily work opens several challenges. First, there are no established user interface concepts to interact with a semantic layer. In the Gnowsis the user interface is spread over several applications. There is the browser, which is used to navigate meta data and open resources. There is also a birdseye view on the system which is implemented as a web application. Resources are annotated with extensions of the applications hosting the annotated resources. To annotate a Microsoft Word document for example, a button has to be clicked which is added as a plugin to the application. The second problem is that everything is a URI. While this is straightforward for files or folders, it becomes more difficult with finer granulated resources. Emails or contacts for example are stored inside one large file — the email application needs to export URIs which identify specific emails inside a large file containing many emails. Another problem rooted in the usage of URIs is that while a URI identifies a resource, it does not locate it. If a file annotated with the Gnowsis is moved it cannot be opened with the browser anymore.

While it is easy to integrate various applications in a Gnowsis environment,

the integrability of the Gnowsis has its limits. It is not possible for example to extend the Adobe Reader to mesh with the Gnowsis. How well the whole framework works was evaluated in a self experiment by the author while writing his thesis. This evaluation showed that the intended goals of the Gnowsis prototype, namely to provide a state of the art proof of concept for a semantic desktop framework, have been met.

The Gnowsis project is superseded by the Nepomuk project [25]. Nepomuk picks up the semantic desktop concepts developed in the Gnowsis and follows them on a far larger scale. It is an attempt to provide a reference implementation for a social semantic desktop standard. This implies that the largest conceptual leap from the Gnowsis to Nepomuk is the introduction of multi user features. The Nepomuk authors identify flaws in related work projects and attempt to improve upon these flaws [25]. The problems they see are:

- lack of evaluation;
- lack of collaboration; and
- lack of integration.

The Nepomuk authors claim that there have been no formal usability evaluations of Semantic Desktop projects. They also say that until Nepomuk no projects provided multi user functionalities. And last, they point out that integration with traditional applications is a problem that has been largely neglected in other projects. It is one goal of the Nepomuk project to address these issues in more detail.

The Nepomuk user interface which is furthest in development is the semantic extension of the Dolphin KDE file browser (cf. Figure 3.13.). Aside from Dolphin there are a few subprojects of the larger Nepomuk which deal with user interfaces. There is a Firefox⁴ plugin for Nepomuk integration of web site bookmarks called Foxtrot⁵. There is also an eclipse plugin for semantically supported software development called PSEW⁶. The last project is the semantic Mandriva online helpdesk⁷. Since these subprojects were at early stages of development at the time of writing of this thesis and there were no scientific publications related to them, they are not considered in this survey of related work. The interested reader is referred to the websites in the footnotes on this page.

⁴www.mozilla.com/firefox/

⁵<http://code.google.com/p/nepomuk-mozilla/>

⁶<http://nepomuk-eclipse.semanticdesktop.org/>

⁷<http://www.mandriva.com/archives/en/enterprise/projects/nepomuk.html>

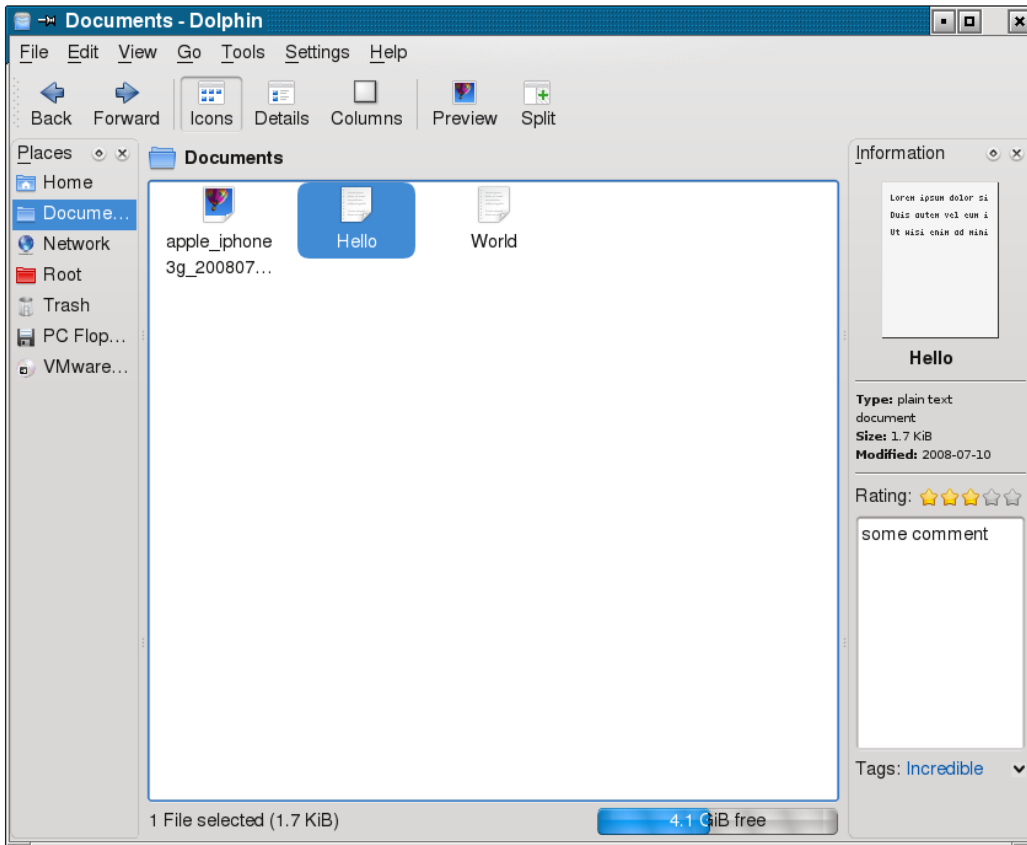


Figure 3.13: Dolphin file manager with Nepomuk extentions.

3.5 Other Approaches

Two of the smaller scale projects are discussed in this chapter. Although the published literature regarding this projects is scarce, and although they did not have direct influences on the Semplorer user interface, they are treated for the sake of completeness.

3.5.1 GLS³

The GNU/Linux Semantic Storage System (GLS³) is one such project⁸. It provides a framework for semantical annotation and retrieval based on these annotations. GLS³ allows its users to tag, relate and annotate files. The main benefit expected from this meta data enrichment are higher precision retrievals. The greatest difference between the GLS³ and SemDAV is that GLS³ does not give up the concept of files. The whole system revolves around enriching a legacy file system. Their application relies on extracting meta data from files. When a file is added to the GLS³ repository the application first tries to recognize the format of the file and to extract any meta data with one of several parsers.

There has been no recent development activity in the GLS³. The developers have published a retrospective analysis of the design mistakes they may have made. They identify two most severe problems the first of which is usability related. The problem they see is that it is not clear where to store files which were imported from legacy media, such as CDs. Data from a CD appears in GLS³ as expected but locating it on the hard drive may be counter intuitive. Since the GLS³ was designed as an extension and not as a replacement for traditional file systems this is a serious issue. The second problem is common to any software development project - the problem of source code decay. The GLS³ was initially developed as a prototype which was extended as the project progressed. Eventually it turned out that unless the prototype was rewritten from scratch no further progress could be made. While this difficulty should be obvious to any experienced software developer it should be kept in mind especially for immature and experimental systems as most semantic applications currently are. The most straight forward solution to this problem is to *plan to throw one prototype away* [13].

⁸<http://www.glscube.org/>

3.5.2 SemFS

The TagFS [11] project is another attempt to escape the restrictions imposed by a tree hierarchical directory structure. The project has been renamed to SemFS [44] without a visible rationale. The crucial idea behind TagFS is to replace directories with tags. The result would be a directory structure without a hierarchy, the order of directories in a path would be irrelevant. Similar to GLS³ TagFS extends an existing file system with meta data. What TagFS does differently is the interface to the user. SemFS tries to be exactly what its name suggests, a file system. A query over the meta data in SemFS is constructed by following a path in a directory structure which appears to be traditional although it lifts several of the restrictions of traditional file systems. Accessing a file which was tagged as “paper” and “WWW2006” is equivalent to accessing either the path /paper/WWW2006 or /WWW2006/paper. This interface is provided to legacy file managers with FUSE⁹, which is a user space file system driver, and WebDAV.

Using a traditional path to navigate an information system poses a great challenge. It is essential to clearly visualize the number of entities which carry a certain tag. This is the only best method to encourage users to use common tags. The number of tags is bound to grow very large; without a filtering based on the number of uses of a tag the navigation through the system becomes tedious and the benefits of tag based retrieval vanish.

Tagging is not sufficient to yield the full benefit of semantic technologies. A tag only ontology does not benefit from the fact that semantic query languages allow predicates which evaluate meta data - it is not possible to query SemFS for all tags which are a number greater than 10 for example. Another neglected benefit is the fact that ontologies allow the construction of taxonomies for concepts. SemFS does not consider the application of hierarchical tags. All these difficulties would be a subject of discussion however if the project had not stalled development. The latest releases of the SemFS/TagFS project are from fall 2006.

⁹FUSE: <http://fuse.sourceforge.net/>

Chapter 4

Usability Challenges in Semantic Applications

In the following we outline difficulties we have met while designing the Semplorer user interface, most of which were also reflected by the evaluation results. The scope of the problems varies – some apply specifically to the sile model and its user interface, while others have a more general validity. All of the identified difficulties can be derived from the attempt to provide a method for communication between users and semantic applications: as with any novel technology, the interface has to be based to the greatest possible extent on concepts with which users are already familiar, while at the same time it must introduce new data manipulation possibilities in order to harness the new technology to its full extent.

4.1 Navigating Large Data Sets

Siles are a semantically enriched version of the file concept, but the sile model does not include a counterpart of hierarchical directories. If we imagine a file system and subtract from it its directory structure, we end up with a number of information entities that easily goes into the tens of thousands [1]. These pieces of information are more often than not inexorably entangled. Hierarchical directories are an efficient method for bringing a certain amount of order into this information cloud, they are however often too restrictive to depict all facets of relations between data. When the large number of objects in a semantic repository are annotated with an even larger number of attributes, organizational means have to be provided that are comparable to traditional approaches in terms of responsiveness, but excel them in expressive power. This challenge applies to any attempt to organize information, although its severity depends on the domain of the application.

Devising a high-performance information storage and retrieval model is much easier for applications with a narrow domain. Such applications benefit from the fact that their users already own a mental model of the information they are working with, which designers can attempt to translate into the application's data model.

4.2 Designing for Diversity

The SemDAV user interface tries to be as generic as possible. It is designed to be a suitable tool for tasks that cope with information organization. Sauer-mann et al. claim that users are unwilling to adopt a generic interface [54], but the success of projects such as Haystack [48] make this claim disputable. When only one generic application is developed, it is possible to experiment with many different extensions at a relatively low implementation overhead. Enabling the development of low cost functional prototypes is important and beneficial at the current, mostly experimental stage of semantic applications research.

The genericity of an application is not limited to technological aspects: it should also encompass the intended user group. An information retrieval application should make as few assumptions about its users as possible. It should address not only scientists from different domains, but also users with varying computer expertise, handicapped users, children and older people. Even users from the same category may exhibit different search behaviors that must be accommodated. An existing information system which fulfills these requirements (next to playing a role in forming a cultural identity and being a social center) are libraries. Stephanidis et al. [63] further elaborate the analogy between libraries and information systems. They argue that the foundational paradigm of user interface design, *knowing the users*, is not directly applicable in applications with a large audience.

4.3 Visualization and Navigation of Ontologies

An obvious way to visualize ontologies is by representing them as trees along the inheritance relationship. This method however is an incomplete depiction of an ontology. A graph G is a tree if it is connected and it is not connected if any edge is removed from G . If a class U is a subclass of both classes, X and Y , which in turn are a subclass of `owl:Thing` then the resulting class hierarchy graph is not a tree. The graph will remain connected regardless which edge is removed. Tools utilizing the tree view circumvent this problem by rendering two nodes for the class U , one which appears as a subclass of X and one which appears as a subclass of Y . This node ambiguity may not be an obstacle for a user who is familiar with the structure of an ontology, but it may be confusing for less experienced users [17].

In Protégé this problem is circumvented by having nodes which appear as children of all their parents. Even this solution however does not visualize relations other than inheritance. While there are methods which attempt to

visualize all relationships (cf. Section 3.2.) these visualizations tend to become cluttered very quickly.

4.4 Visualizing Queries

The query language in the Semantic Web is the SPARQL Protocol and RDF Query Language (SPARQL). Similar in spirit to SQL in relational databases, it is designed for developers, not for end users. Ever since command line user interfaces were replaced by graphical user interfaces there have been efforts to design intuitive query interfaces. Query languages such as SQL and SPARQL rely on boolean algebra for their semantics. Queries written in these languages are also called boolean queries because they can be translated into a boolean formula which carries the same semantics. This generalization plays an important role in visualizations. The reason for this is that methods developed for one query language can be applied to any other boolean query language.

There have been several efforts to visualize boolean queries [8,29,68]. However most are domain specific and make stringent assumptions about their application. What the literature agrees upon is that a user interface providing the full expressivity of boolean algebra is difficult to handle by lay users. This is why the visualization of boolean queries is always a tradeoff between intuitivity and expressivity. On a closing note, it is interesting to observe that even today's leading search engines on the web avoid tackling this difficulty. They provide user interfaces akin to the command line interfaces of past decades. Studies have shown that their boolean algebra features are very rarely used [28,61].

4.5 Semantics of Resource Names

The relation of a name to the thing it names is both, problematic and well researched [32]. Although this difficulty has been recognized in several leading semantic information projects [30,35,53], it has not been explicitly addressed. In the Semantic Web this problem stems from the usage of URIs. A URI does not have to be human readable and it usually refers to some object. This means that there is a human readable representation of the URI, the URI itself and an external resource which are all in some semantic relationship.

There are several theories that attempt to model these relationships between symbols, concepts and concrete things, all of which agree upon the idea that a symbol and its referent are two different things. Ogden and Richards have introduced the semiotic triangle as a model of meaning (Figure 4.1). Their

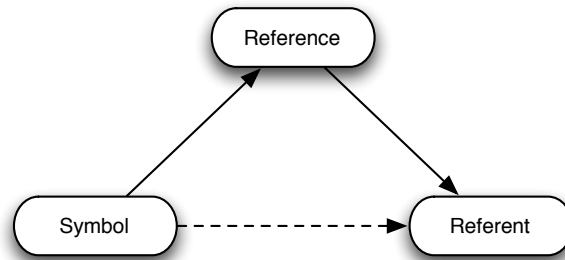


Figure 4.1: Semiotic triangle [42].

triangle expresses the idea that a symbol refers to a referent by the means of a reference, or, more concretely, an expression in a natural language uses a thought or an idea to refer to an object. When someone uses the word “horse” she is communicating a thought which entails a possibly real horse. The exact way of how the angles of the triangle refer to each other has been subject of a very large scoped linguistic and philosophical research.

The semantics of the sile model, and the meta models of the Semantic Web as a whole, fit well into this construct. The symbol is represented by the human readable name of a URI and the referent is the URI which refers to the resource itself. The only flaw in this translation is that the resource pointed to by the URI does not necessarily exist – it is possible that the meaning of the URI is entirely stored in the meta data describing the URI. Thus an optimal reference model for the Semantic Web has yet to be devised.

Even this incomplete model shows that the relation between resources and their names is of interest to a user interface designer. This edge of the triangle is the bridge between the mental model of a user and the machine representation of the semantically enriched data. A reference without a human readable name would be of little value.

If we consider resources as concepts in the linguistic sense, it is natural to assume that URIs and their names should not have implications on each other. In a natural language the word “horse” can refer not only to a specific horse but to every animal of that species. Following the analogy of natural languages, several URIs can thus have the same name, and one name can symbolize several URIs.

One question which arises is whether two things denoted by the same name have anything in common. There are two competing reference theories which answer this question differently; the theory of realism and the theory of nominalism. The former assumes that two things bearing the same name share more than just their name. In its most simple, traditional form the theory of nominalism assumes that the only thing referents have in common are their

names.

The ambiguity of meaning has been treated only very recently for Semantic Web issues by Garcia et al. [23]. Their research however only deals with synonyms in ontology mappings. The author of the thesis at hand is not aware of any works treating problems of ambiguous labels for semantic resources.

Putting this problem into a context of user interface design complicates the matter even further. User interfaces often rely on metaphors to bridge the communication between humans and machines. The meaning of metaphors however is often not congruent with the intention of the designer, a metaphor may have unintended implications. It has not been researched what implications metaphors may have on the semantic relation between human readable names and URIs.

Another issue is that information stored in a computer is usually intended for manipulation. Manipulation gestures which come into play need to be consistent with the metaphor visualizing the information on the one hand and with the meaning of the gesture in other applications on the other hand. For example dragging and dropping an element onto another element is an appropriate gesture for putting the two elements in a semantic (as in Semantic Web) relationship. However relationships in the Semantic Web are directed, this means that dropping one element onto another can have one of two semantics: either element A is in relation to element B or vice versa. Which of these meanings is the more natural one is an open question. Another open question is what other interaction concepts have ambiguous or unprecise meaning in the context of semantic information and what sort of impact these imperfections may have.

4.6 Introduction of New Vocabulary

Semantic technology is described in a domain specific terminology. The vocabulary used by semantic systems researchers is only understood by semantic researchers. One of the reasons is that many of the terms have been adopted from other fields of research but their meaning was changed. The word “ontology”, for example, originally meant “the study of being”. In the field of Semantic Web research however an ontology is best described as a special form of a taxonomy, or as a formalized conceptualization.

It can be assumed that the majority of the users of semantic applications are not familiar with the language used by the scientists who created the application. An average user will be hopelessly overwhelmed by a message saying that the ontology currently in use does not allow the class *Person* to have a $1 : n$ cardinality for the property *name*. The user’s situation would be

even worse if she had background knowledge in philosophy.

In cases where a system has a well-defined application domain an obvious way to circumvent this wording problem is to reformulate messages: For example, the application might translate the message to “A person may only have one name”. Downey has recognized this problem in the evaluation of the SEEK user interface [17]: during the evaluation one of the test subjects noted that the term “annotation” was too overloaded and thus not appropriate. If the application however targets more than one domain the approach of explicit wording becomes infeasible.

4.7 Addressing of Challenges in Related Projects

After reviewing the largest semantic desktop projects and outlining the challenges faced in the design of these applications, it is instructive to look into how these challenges are addressed. Figure 4.2 gives a rough overview of the situation. This table however is a projection which hides information required to interpret it objectively. It does not show for example that Gnowsis, KDE4, PSEW and the Mandriva helpdesk are all closely related – they are subprojects in the larger Nepomuk project. In the following a more detailed analysis of the relations between projects and challenges is given.

4.7.1 Navigating Large Data Sets

The challenge of navigation is addressed in the Lifestreams project following the chronological paradigm. The user can navigate back and forth in time and dial to a point of time in the past or in the future. A mechanism supposed to support the navigation process are so called *substreams* which are basically the same as the main stream except that they only display data which matches some user supplied criteria.

In Haystack the interface support for navigation is minimal. Users are limited to scrolling lists. The idea in this project is that the appearance of the lists is highly customizable. The authors of the project also heavily rely on automatic and semi-automatic classification of new information. They provide simple facilities for information classification – information can also be classified in more than one class.

Treating the Nepomuk project is a little more complicated because there are several user interface efforts in Nepomuk. The Gnowsis project is also mentioned here because although it is a project in its own right it is the project from which the larger Nepomuk spun off.

	Lifestreams	Haystack	mSpace	Gnowsis	KDE4	PSEW	Foxtrot	Mandriva Helpdesk
Navigating large Data Sets	♥	♣	♥	♣	♠	♣	♠	♥
Designing for Diversity	♠	♣	♠	♠	♠	♠	♠	♣
Visualization and Navigation of Ontologies	♠	♣	♠	♠	♠	♣	♣	♥
Visualizing Queries	♠	♠	♥	♠	♠	♠	♠	♥
Semantics of Resource Names	♥	♣	♠	♣	♠	♠	♠	♠
Introduction of New Vocabulary	♥	♣	♠	♠	♠	♠	♠	♥




	Not Recognized		Recognized and not Addressed		Recognized and Addressed
---	----------------	---	------------------------------	---	--------------------------

Figure 4.2: Challenges and Projects.

In Gnowsis the navigation is realized with a modified Wiki. The Wiki is used to render the relations between and classifications of data. This rather minimalistic approach was chosen because the resources in this project were very limited. There are four other Nepomuk subprojects which provide a user interface to semantic data: the PSEW eclipse interface, the KDE4 file manager, a Mozilla Firefox named plugin named Foxtrot and the Mandriva semantic helpdesk. Data in PSEW is navigated using simple scrolling lists and tree views. Dolphin, the KDE4 file manager, does not provide any interface for navigating the semantic annotations of files. This is also true for Foxtrot, the Firefox plugin.

The matter is a little more complicated for the Mandriva helpdesk. To this time there is no publically accessible user interface or a description thereof for this project. However, the project will officially continue for a few more months so the roadmaps in the project deliverables will be used to infer the characteristics of the finished user interface. For navigating questions and answers they propose a zoomable map based on iMapping [citation]. For the navigation of the RDF graph they suggest using faceted browsing.

Most interesting with respect to this challenge is the mSpace project as its main concern is navigation. They suggest a quite flexible version of faceted

browsing in which one can swap, add or remove facets. Further details on the exact workings are covered in Section 3.4.2.

4.7.2 Designing for Diversity

Most of the projects covered in this chapter are not limited to a specific user domain. They have the implicit ambition to be a generic user interface for information management, in Haystack's case, or for navigation, like mSpace for example. However none of them have addressed the fact that an application with a generic purpose will be utilised by users with different levels of skill or maybe even physical impairments.

Some projects document this challenge in their publications without proposing a solution. In the Haystack project for example, the authors say that it cannot be expected that every user will work with the same schema. They propose providing flexible user interfaces which can be tailored by users as needed. There are two problems with this proposition: first, the authors see that users will differ but they see the difference in the requirements, not in the capabilities of users. They do not document expectations that their software might be used by children, elderly people, people in non scientific environments and how they cater to these user groups. The second problem is that they suggest having solved the problem by providing a user interface which can be customized. They explain this with a metaphor [27]: software development and user interfaces are like carpentry and interior design – people may not want to build their own furniture but they want to have a say in how the furniture will be arranged in their dwelling. They miss however that very few people are competent interior designers. A user may be able to tell when an arrangement suits their needs but it cannot be expected that average users can make good user interfaces.

The projects under the Nepomuk banner largely neglect this issue with one exception: the Mandriva helpdesk¹. In their project documentation they have meticulously researched their user population and discovered that their platform will be used by users with very different backgrounds. They make several conclusions derived from this fact, it is however unclear how this recognition will impact the appearance of the user interface.

In the mSpace project this issue is not addressed explicitly. It is an interesting paradoxon because the authors are well aware of the large user population they are targeting. They make large scale evaluations to ensure the usability of their product, however the users in their experiments are mostly students who are a very specific group of users.

¹<http://www.mandriva.com/archives/en/enterprise/projects/nepomuk.html>

4.7.3 Visualizing Queries

The problem of intuitive query visualization and manipulation was tackled only by one project – mSpace. The Lifestreams project provides an input field for entering textual boolean queries. The remaining projects provide full text searching and sometimes the possibility to enter SPARQL queries. In mSpace queries are visualized very unobtrusively and intuitively: the contents of a column are the resources which satisfy the conjunction of the column headers to the left. This intuitivity however comes at the price of reduced expressivity.

4.7.4 Semantics of Resource Names

The issue of putting human and machine readable resource identifications into a semantic relation is interesting in that it is recognized and covered in literature [30, 52] but authors mostly leave at that and withdraw from an indepth discussion. Impacts of this challenge cannot be found in the user interfaces of related project. One exception is the Lifestreams project, there they recognize that forcing users to give unique human readable names to every file is a difficult and mostly unnecessary for users. They resolve this issue by shifting the unique identification to a time axis rather than names. The reasoning is that chronological order elicits correct contextual associations in users on the one hand and it is a unique identifier for machines on the other hand. One issue they have not addressed is that in a colaborative enviroment the time axis can become rather crowded and using time stamps as the main identification may become insufficient.

In Haystack the issue of naming resources appears very peripherally. In section 4.2 of the paper “Haystack: a User Interface for Creating, Browsing and Organizing Arbitrary Semistructured Information” [47] the authors recognize that users prefer human readable names over URIs:

Perhaps the most basic view of an object is a simple reference to it on the screen by a human-readable name. The ultimate fallback name for any object is its URI, but URIs tend not to be meaningful or memorable for humans. Instead, if `dc:title` or `rdfs:label` properties are provided, Haystack will use one of the values of these properties, giving higher priority to `dc:title`.

They propose deriving a human readable name from commonly seen RDF properties. However they neglect synonyms, ambiguities and other difficulties arising from building the bridge between human and machine readable representations.

In the Gnowsis project this difficulty is treated very curiously ([52] Section 3.5.2). The author names the section dealing with this topic “The URI Crisis” and briefly covers the problem of semantics in philosophy. It is not clear however what impact this discussion has on the workings of the Gnowsis application. In the context of the Nepomuk project the philosophical debate is continued [35] however the direction is changed. In Nepomuk the topic of discussion is no longer semantics as such but the old question of whether computers can think and how this relates to mental models of users. In the opinion of the author of the thesis at hand the question of whether machines can think is irrelevant (cf. Section 2.1.). The irrelevance of this question makes its treatment unsuitable as a foundation for a mental model of semantically enriched information.

4.7.5 Introduction of New Vocabulary

The introduction of unfamiliar vocabulary is treated differently in the reviewed projects. While the Lifestreams project is not a semantic desktop application, they too introduce new vocabulary. They have opted to introduce words like *streams* and *substreams* to trigger helpful associations in the minds of their users. A stream refers to the stream of time, containing all the user’s documents. A substream is a part of this larger stream. In the mSpace project the usage of unfamiliar concepts is omitted entirely. That way there is no need to introduce new vocabulary. The Haystack project takes an intriguing approach: the authors recognize that users may not be familiar with terms like RDF or OWL and they propose the RDF manipulation language Adenine to alleviate this problem [46]. In the opinion of the author of the thesis at hand, introducing a textual notation which is even more expressive than the previous one does not address the challenge of new vocabulary appropriately, it merely shifts it. However there is no evidence either in support or against the proposition of the Haystack team. The Gnowsis and the Nepomuk projects do not address this issue at all.

Chapter 5

Application of Interaction Design Guidelines in the Development Process

This chapter outlines the design process of the Semplore application. The process followed largely resembles the one proposed by Jakob Nielsen and outlined in Chapter 2. Let us briefly review the steps of the process:

- Know the user.
- Competitive analysis.
- Setting usability goals.
- Parallel design.
- Participatory design.
- Coordinated design.
- Heuristic analysis.
- Prototyping.
- Empirical testing.
- Iterative design.
- Collect feedback from field use.

Knowing the users was taken into account but it is difficult to apply this concept directly to the potential users of the Semplore. The difficulty is that the user group is not from a well defined domain – anybody who uses a computer to store personal information should be able to use it. This includes administrative staff as well as experts but also children, old or handicapped people. This problem is elaborated in more detail in Section 4.2. The knowledge about users was derived from research of the World Wide Web. While this may be an inaccurate approximation, there are several parallels between tasks on the web and on a Semantic Desktop which justify this juxtaposition. Both, the web and the semantic desktop are, or at least were intended, as large scale information systems. Some of the most common tasks done on both systems

are search, retrieval, storage and manipulation of information. More details on the parallels between the semantic and the traditional web can be found in Section 2.1.

A competitive analysis was performed meticulously for the Semporer, however the results were very limited. At the time of the writing of this thesis, Semantic Desktop user interfaces are mostly developed in academic settings, there are no comparable commercial products. Even the few academic projects are at an experimental level. Three projects were identified as potential competitive products: Lifestreams, Haystack and Nepomuk [20, 25, 27]. The Lifestreams project was abandoned many years ago and neither the application nor a suitable platform is available for trial. Scarce documentation, cryptic error messages and later the unavailability of a download prevented a thorough evaluation of the Haystack project. When the Semporer development commenced, the large Nepomuk project was in the final stages of its publically funded period. At the current stage their user interfaces are rudimentary at best. Further details are covered in Chapter 3.

The next step is about identifying the usability goals in order to achieve a compromise between contradictions and consistency in the user interface. The ultimate goal of the SemDAV effort is to eventually replace traditional file systems. The less ambitious goal of the Semporer however is to serve as a flexible prototyping platform for experimenting with as many different interaction paradigms as possible. Because file managers are not intended for accomplishing complexly interwoven processes, it is possible to keep the user interfaces for different tasks in the Semporer independent from each other. The interface for retrieving siles for example does not interact directly, not even on a source code level, with the component for visualizing siles. As long as all components agree on the imposed drag and drop interaction “language” there are practically no contradictions between interaction concepts.

The next three points of the process are not isolated steps but rather guidelines for the design phases of the project. The first suggestion is to parallelly design several versions of an application. Since the Semporer is an experimenting platform this proposal was taken particularly seriously, not so much by experimenting with many different versions of the Semporer but by supporting parallel prototyping within the Semporer. With its loosely coupled components and unified interaction paradigm the Semporer caters well to different developers working on different versions of the same component without replicating any work. Since the resources to formally evaluate the Semporer were not present the finished application does not abandon the parallel prototypes – there are different versions of retrieval and visualization components which users can choose from.

The second suggestion is to involve users in the design process. Because of the lack of a clear user domain this suggestion could not be applied in the design process of the Semplore. The considerations of the user domain are further elaborated in Section 4.2.

The last proposition concerning design is to coordinate the design process between designers in order to achieve a consistent user interface. This recommendation was strictly adhered to in the development of the Semplore. A single person coordinated the general architecture of the Semplore providing a framework for other developers to work in. This framework includes a communication mechanism to allow components to communicate with each other as well as an application wide drag and drop mechanism. Further details are covered in Chapter 7.

The next phase in the process is the evaluation phase. Nielsen specifically suggests a heuristic evaluation because of its low cost and easy of application. The heuristic analysis of the Semplore is covered in Chapter 8.

Further it is suggested that the development is prototype driven. The development of the Semplore was heavily prototype driven – starting with a first prototype made of paper (depicted in Figure 5.1), through a prototype written in SmallTalk (cf. Figure 5.2.) up to the last Semplore version, which itself is a prototyping platform.

Although the Semplore was evaluated using a heuristical method, a full scale usability survey remains out of the scope of this thesis. The empirical testing point remains largely neglected.

It is not sufficient to built prototypes, the development of prototypes has to be iterated. Iterations in the Semplore project were mostly due to improvements from a technical perspective. The Semplore was conceived as a part of the SemDAV project which aims to provide a communication framework for a semantic desktop. As such larger attention was payed to improvements behind the scenes than at the surface of the application.

Since the Semplore was never intended for actual deployment, the perceiving of its unfinished state is not endangered by a delivery date. In other words: feedback from the field was not collected since the Semplore never was in the field.

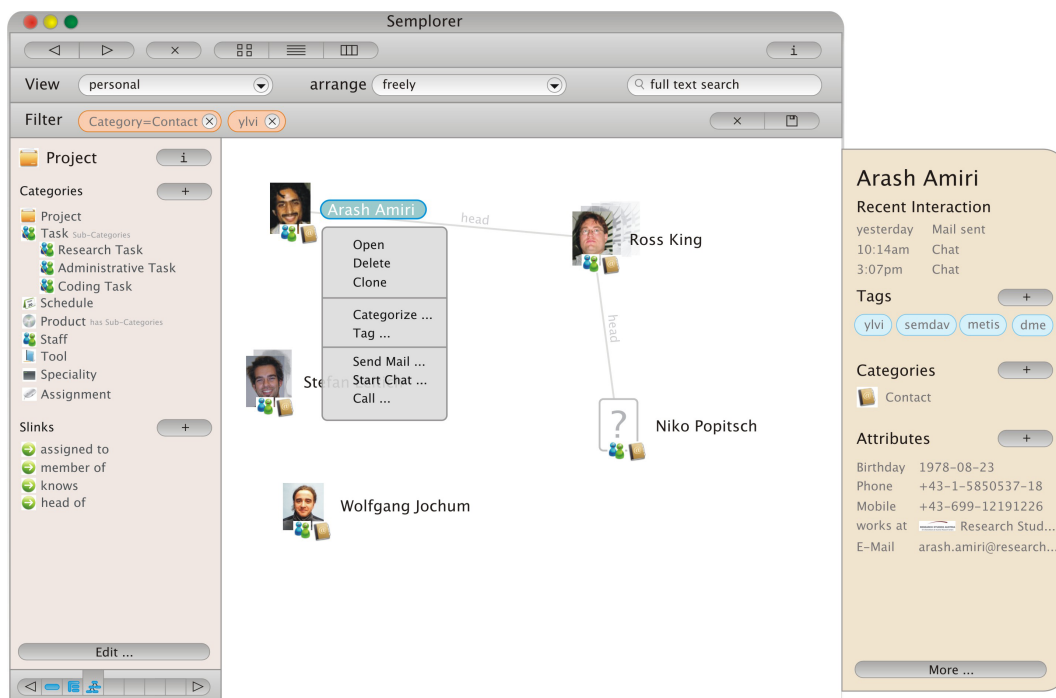


Figure 5.1: Paper mockup of the Semplorer.

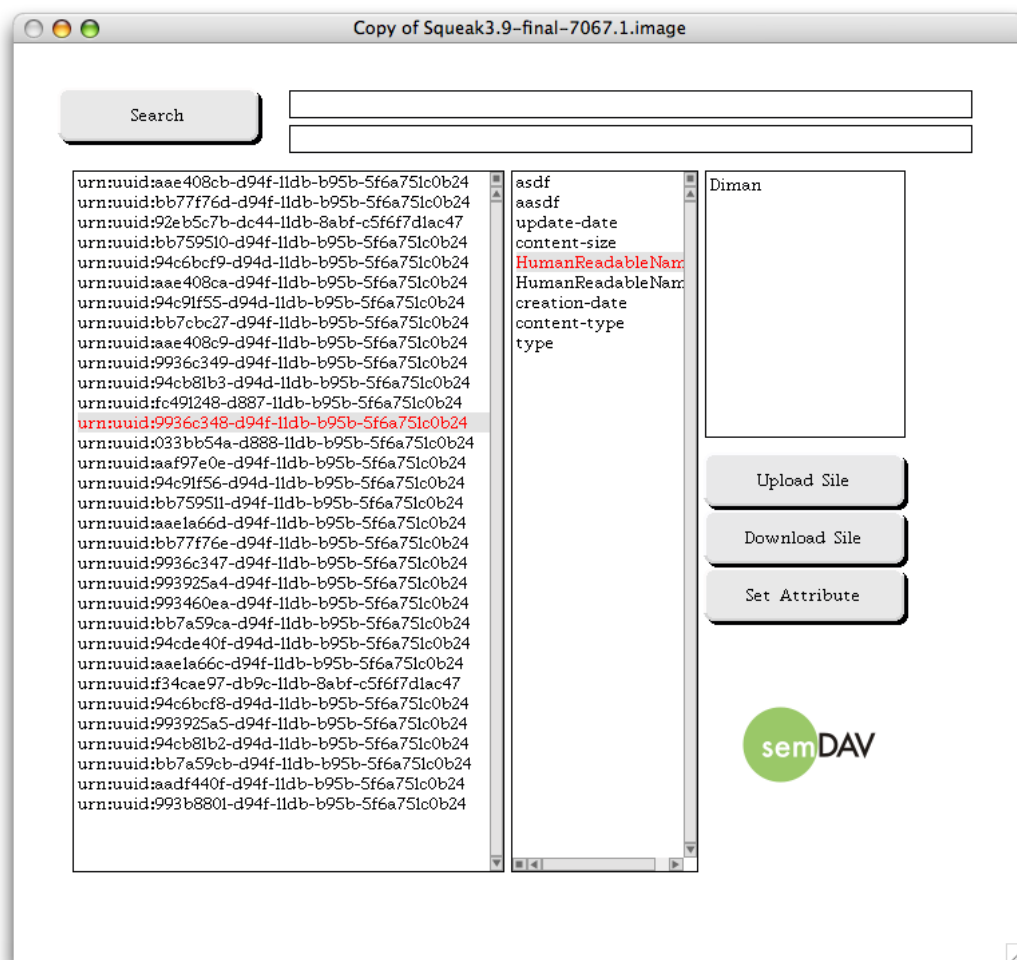


Figure 5.2: Early Smalltalk prototype of the Sexplorer.

Chapter 6

User Interface Design

The goal of this chapter is to introduce the rationale behind aspects of the design of the Semporer. To understand the design decisions leading to the Semporer it is necessary to keep in mind, that the Semporer is a byproduct of the SemDAV project. The central goal of this project was to develop a protocol for the integration of a collaborative semantic desktop environment. The role of the Semporer was that of a prototypical client used to identify requirements and feasibility of protocol features. As such several of the best practices of user interface design were not applicable. The Semporer was used as a platform for exploring interaction concepts towards a semantic desktop. This is also reflected in the highly modular software design, all user interface components were designed to be easily replaceable so that the overhead of rapid prototyping is kept at a minimum.

Because of the limited amount of resources available to the authors of a thesis like the one at hand, not all of the identified challenges could be addressed in depth. Particular attention was paid to methods for navigating large data, visual boolean query interfaces and ontology visualization for end users. The treatment of these challenges is treated in the following sections. Further, the challenge of introducing unfamiliar vocabulary was touched upon although not beyond the providing of a potential approach. The question of the semantics of resource names was considered [64] but the analysis of its relevance and impact on semantic applications needs further investigation. While the Semporer was designed with a diverse user population in mind, no formal evaluation in this direction was undertaken. The topic of a diverse user population is part of the future directions of the SemDAV project.

6.1 The Example of Bob

In order to demonstrate the interaction mechanisms of the Semporer the fictive person Bob is introduced. Bob is a provisional persona as described by Cooper et al. [15]. The persona is not the synthesis of empirical data, rather he is a stereotype created to serve as an example for Semporer interaction. It is important to note that the persona and its assumed behavior was not used to justify design decisions. Because of the lack of the foundation on empirical data, Bob is entirely unsuit to serve as a design tool.

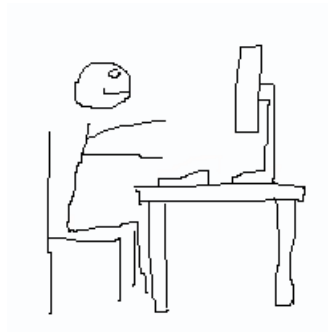


Figure 6.1: Bob.

The literature [15] suggests that fictive personas should have pictures. This enforces the empathy towards the persona. Bob is an entirely fictional persona constructed without even reliance on stereotypes. His depiction (Figure 6.1) reflects this sketchiness.

Bob is a student. He attends Neverland university where professors pace courses in sync. This semester Bob is taking the theoretical computer science class and the introduction to compiler development class. The synchronous pacing means that for example when the professor teaching the theoretical computer science class introduces regular expressions, the compiler development professor is teaching about lexers.

In the following sections descriptions of Bob doing everyday tasks with the Semporer are used to support the understanding of the interaction concepts in the Semporer.

6.2 Overall Appearance

The user interface of the Semporer is modelled after traditional file browsers. From today's perspective it is difficult to say how the widely known layout of file browsers developed. It is however easy to observe that practically all common directory navigation tools follow the same paradigm: on the left there are starting points into the directory structure; the middle pane displays the current context – the files and folders in the current directory; when present, the pane on the right side displays details about the currently selected file; and finally, at the top there is some sort of visualization of the current working directory. The three most common file managers are depicted in Figure 6.2.

The parallels in the Semporer interface are made apparent in Figure 6.3. The Semporer interface is divided into 5 panes. The first pane, the middle one, selects the siles residing in the current context. The second part, is usually

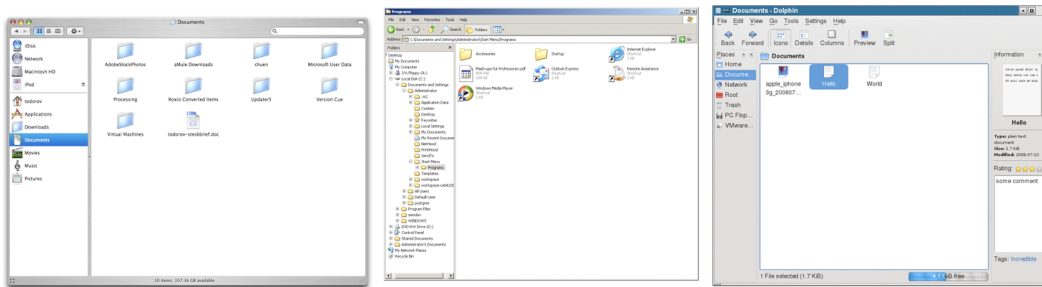


Figure 6.2: Apple Finder on the left, Windows Explorer in the center, KDE Dolphin at the right.

not found in file browsers. It plays the role of an explicit clipboard. Entities which are used often can be placed there for later reference. The pane on the right (numbered 3), displays, like in traditional interfaces, details about the selected file. Number 4, at the top, is the equivalent of the current directory – it is a filter describing the working context. And finally, the fifth part on the left, is where usually some starting points into the directory structure can be found. Because the Semplore does not operate on directories, the equivalent categorization tools have been placed there, namely specs and tags. In the following sections the navigational components of the Semplore are discussed in more detail.

The main interaction gesture in the Semplore is dragging and dropping. This gesture is maybe the most important one in the direct manipulation user interface paradigm. Ben Shneiderman, a recognized pioneer of direct manipulation interfaces, has collected inspiring quotes of user interface researchers who value the concept [60]. All of them agree, that having an explicit and tangible visual representation of an interface makes not only for a flat learning curve but also for a user interface that is fun to use.

In the following a few common interactions with the Semplore are described on the example of Bob the student.

When Bob wants to add a new document to his SemDAV space, he simply drops it onto the file pane. This uploads the document to the central repository and adds some common attributes like a creation date, a document name and a few others. To annotate the file, Bob can drag tags from the tag pane or categories from the spec pane and drop them onto the file. He can either drop them on the file rendering in the file pane or in the detail pane, both targets have the same effect. When dragging over a valid target, the target lights up with a highlight to notify Bob that he can drop whatever he is dragging there.

When Bob is frequently using the same tag, he can drop it into the file pocket for quick reference. The file pocket is a visual clipboard – a temporary

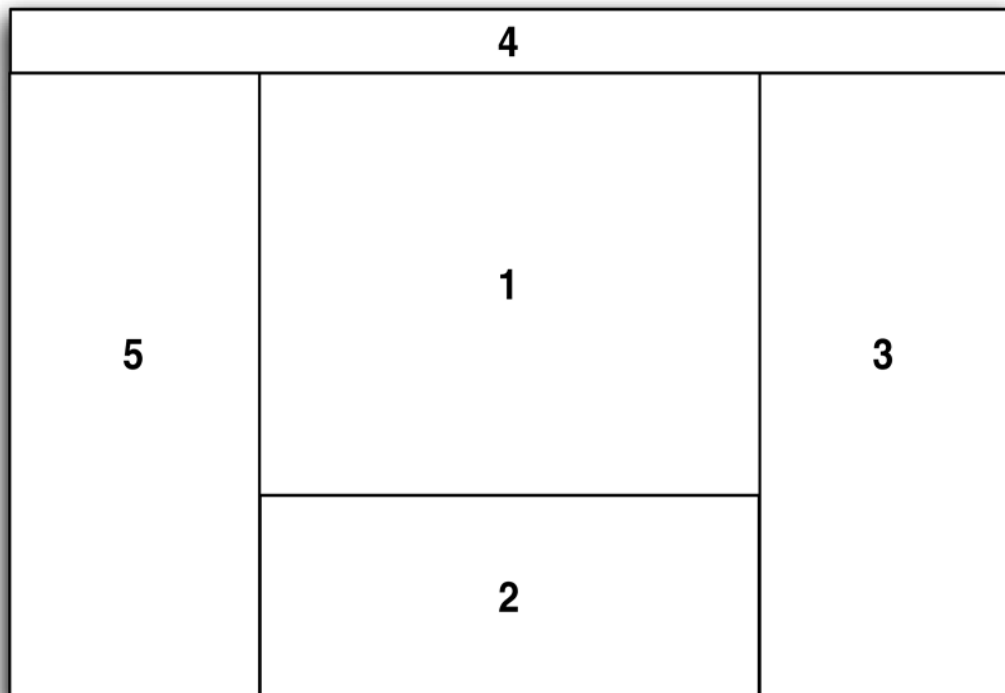
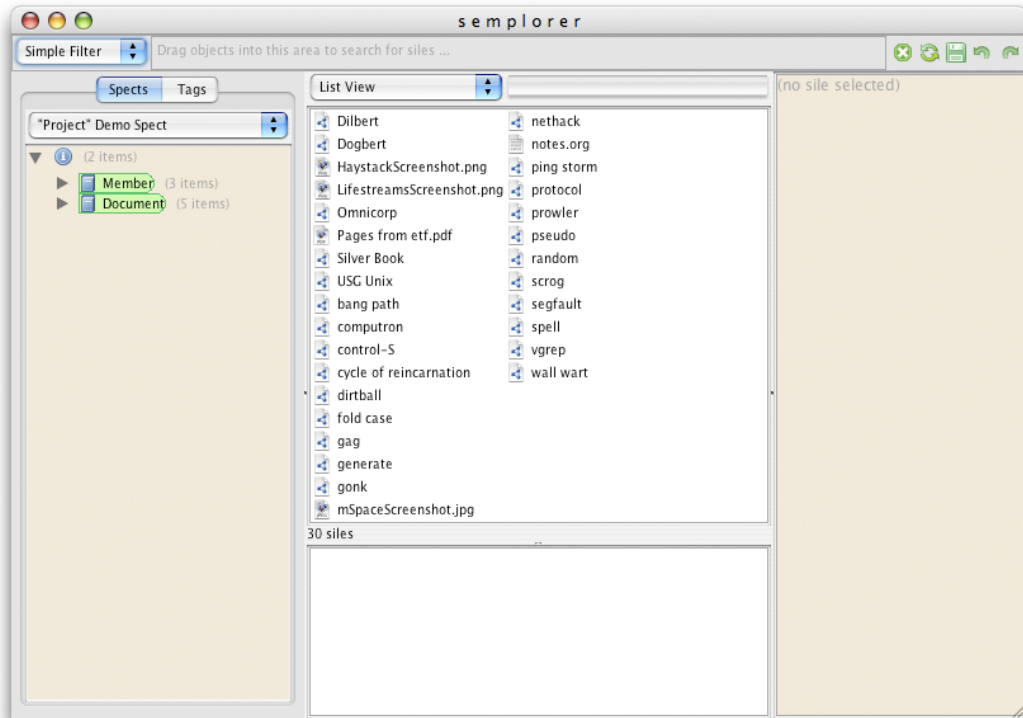


Figure 6.3: User interface components of the simplorer.

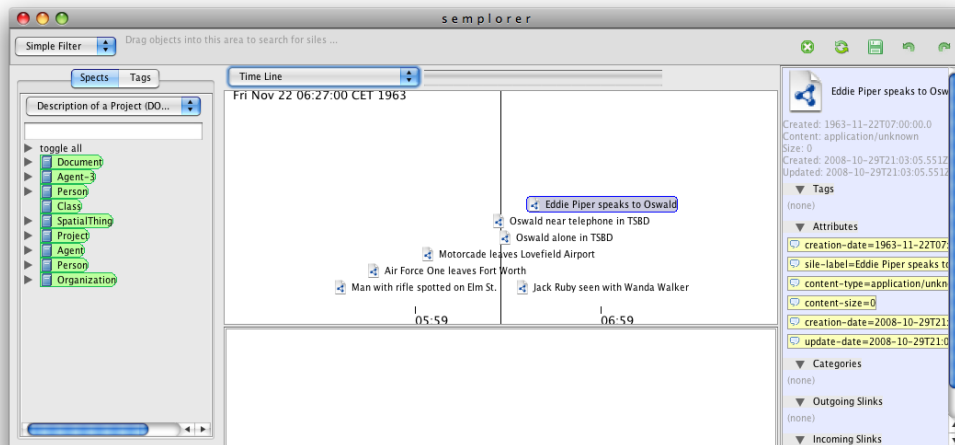


Figure 6.4: Screenshot of the timeline view in the Semplorer.

storage for quick access of documents or annotations which are used frequently. To open a sile Bob, only needs to double click it – just like in any other file manager.

The most complex semantic annotation is the slink. To add a slink from one sile to another sile, Bob drags the first sile and drops it onto the second. He is then presented with a pop-up menu displaying all slinks which can be placed between the two siles. Slinks can be followed similarly to links on the web. When Bob navigates to a sile, he can see the incoming and outgoing slinks in the detail pane. If he clicks on one of the target siles of an outgoing slink, the Semplorer will change its focus to this target sile.

6.3 Navigating Large Data Sets

In this section a chronological information navigation mechanism is proposed, which is inspired by the visualization in the Lifestreams [20], and TimeScope [50] (see also Chapter 3.) projects. The contribution of the proposed method is an alternative application of the vertical axis of the user interface. Instead of neglecting it, as in Lifestreams, or using it to display contextual vicinity, as in TimeScope, it is used for the implementation of a zooming method which does not actually shrink the size of the displayed siles. Let us first elaborate the mechanics of the Semplorer timeline visualization on an example.

This week Bob’s homework for the compiler related course is to build a parser. After reading the assignment, he realizes that parsers are just a practi-

cal application of context free grammars. Unfortunately, Bob cannot remember when context free grammars were covered in the theoretical course. However, he roughly remembers when parsers were covered in the compiler lecture. He opens the Sexplorer and quickly crafts a filter¹ which shows him only siles related to either of the two lectures. Bob opens the timeline view and pans it roughly to the correct part of the month in which he remembers hearing the parser lecture. Then he zooms in until he can make out what he has stored in the repository around the time. Finally, he discovers a sile labelled as “*context free grammars*”. He clicks on it to inspect it more closely and sure enough, it contains the notes he was looking for.

Bob is a forgetful student. He often forgets to start doing his homeworks on time. To alleviate this problem he places reminders in his SemDAV space. To do this, he simply adds a “reminding date” attribute to the sile containing the homework assignment. The reminding date is the time at which he would like to start doing his homework. After attaching the attribute, the homework sile will be placed at the reminding date in the timeline.

Most of the ideas in the timeline view are not new. The idea of computer aided time centered organizing paradigms goes back to Freeman’s and Gelerner’s Lifestreams [20] (see also Section 3.3.1). The idea of making reminders appear in the future for example is directly inspired by their work. The main difference between the Lifestreams and the SemDAV timeline is that SemDAV not only introduces an ordering but actually places siles on a timeline. Short time intervals between siles are mapped to short distances on the screen and long intervals are mapped to long distances on the screen.

The main problem this method introduces is, that intervals in which many siles were created and intervals in which few siles were created are not displayed equally efficiently. In a period where new siles were rare a lot of the screen space is used to display time in which nothing interesting has happened. On the other hand, in periods of high activity the screen space may not be enough to display all siles while maintaining true relative distances. At the core of this difficulty is the fact that sile dates are not spaced with uniform density. To solve the problem an efficient method for visualizing dense clusters has to be devised.

One of the most common methods for visualizing dense clusters are zoomable interfaces. Zoomable interfaces however challenge the designer with two difficulties. First, the design has to entail keeping the user in context. When changing zoom levels it is easy to get lost. For this reason transitions between different zoom levels have to inform users about how the new zoom level is a mapping of the old one. The second challenge is that of detail abstraction.

¹see Section 6.5 for a further explanation of filters

The higher the zoom level, the more information is shown on the screen. With rising information the importance of details sinks. The whole point of zooming out after all is to provide a larger picture. When zooming out a picture for example, the resolution shown on the display becomes smaller.

Two mechanisms ensure that the information context is not lost when panning and zooming the timeline. The first mechanism restricts magnifying only to the center of the display. If you want to visualize a very dense cluster, you first pan it to the center of the screen and then change the zoom level. This works a lot like one would go about zooming in and out on a microscope – you first put your object of interest under the lense and then change the distance between the lense and the object to achieve the desired zoom level. The second method keeping the user in context are smooth transitions. Like in the microscope analogy, any zoom level is allowed so that keeping a mental map of the observed objects is intuitive.

The problem of abstracting information is tackled in a slightly unconventional way. If two siles on the timeline are so close to each other, that they cannot both be displayed without overlapping each other, the later sile is stacked on top of the earlier sile. When zooming out, one sile is stacked on top of others so that a tower of siles is built which grows towards the top of the display. When zooming in on the other hand, the tower is taken apart until all siles sit next to each other in chronological order.

There are several possible future directions the SemDAV timeline visualization could follow. Additional functionality can be introduced to allow sorting not only by date but by arbitrary attributes. Although this idea already appeared in the context of the Lifestreams project, little has been published on its usefulness. Other mechanisms worth exploring are visual effects such as blur or variable focus to support the stability of a working context. Another idea would be to introduce a time scale which is not equidistant but logarithmic, so that longer time intervalls appear shorter if they are further away from the center of the display. And finally, it is worth investigating wether the stacking of siles on top of each other is a sufficient abstraction method or wether it needs further refinement.

6.4 Visualization and Navigation of Ontologies

Findings of the comparative study performed by Katifori et al. [2] (also reviewed in Chapter 3) played an important role in the design of the *spect* view in the Sexplorer. Two concrete implementations were implemented both of which follow the mechanics of a tree view.

The first version of the *spect viewer* (see also Figure 6.5) displays all children

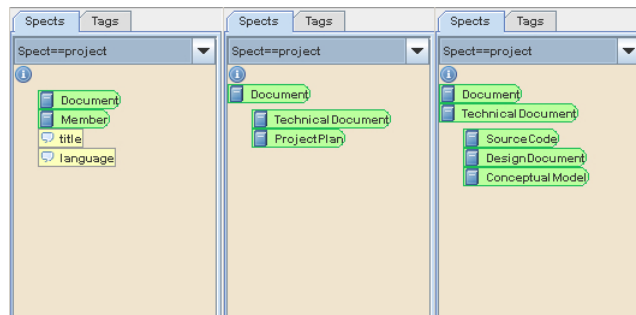


Figure 6.5: Browsing spect.

nodes of the current node slightly indented. Beyond the current node a trail of already visited nodes is displayed. Every time the user clicks on a child node, the viewer navigates to the node and expands its children. If the user clicks on a node in the trail, the viewer steps back to the selected node. Figure 6.5 shows three spect viewers showing different hierarchy levels of the same ontology. The viewer on the left shows the spect as it initially appears to the user. The second pane shows the state after the user has clicked on the *Document* category—the category is now in the trail, and the viewer displays the sub-categories of *Document*. The third viewer shows the next level; here a trail of two categories has been walked and the sub-categories of *Technical Document* are displayed. Clicking on the “i” symbol resets the viewer back to the root node. If the interface is in the state depicted on the right and a user clicks on the *Document* node, the interface traverses up the hierarchy to the node representing the *Document* class, as shown in the middle screenshot.

The results of the usability evaluation however have been divergent with regard to this approach. The two problems pointed out by evaluators were firstly that the parent/child relationship is not made sufficiently explicit, as e.g. the relation between the categories *Document* and *Technical Document* is not clear from the visualization. Secondly there exists no “bird eye view” of the ontology: searching for a particular category is difficult because the user has to remember the trail she had to walk to reach it.

These problems were addressed in the second iteration of the spect view implementation. This version is described on the example of Bob. In order to retrieve lectures with the filter interface, Bob needs to annotate them. To do this, he is using an extensive ontology designed specifically to describe Bob’s curriculum. After a lecture Bob drags his lecture notes into the Semporer and then he classifies them by dropping categories on the newly created sile. He categorizes lecture notes in theoretical computer science by dropping the appropriate category on top of the sile. However, since the spect contains all

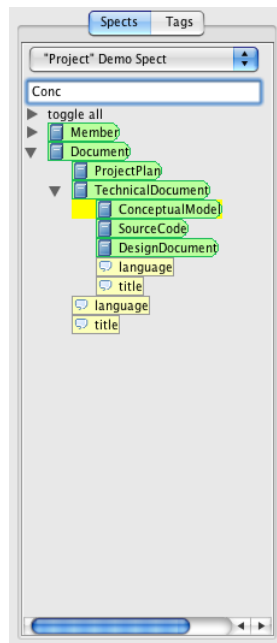


Figure 6.6: Second iteration of spect view.

lectures in Bob’s curriculum, it is a little unwieldy. Browsing the spect every time Bob wants to categorize a document may soon become tedious. When Bob knows the name of the lecture he can type the first few characters in the text field above the spect view. After every character the view is updated to expand and highlight only the matching categories. After typing three to four characters, the theoretical computer science category is the only one that remains highlighted, now Bob can swiftly drop it ontop of his notes.

The design of this visualization method has its roots in the results of the comparative study performed by Katifori et al. [2]. The authors of the study conducted preliminary usability evaluations of different ontology visualization methods and found that a simple tree view is the most effective tool in practice. They evaluated the ontology tree view in the ontology editing application Protégé². This particular implementation was found to have some flaws: expanding and collapsing of nodes works only when users click on the arrows beside the node, clicking on the node itself has no effect; there is no effective method to search for a class when a user recalls its name but not its location in the hierarchy; there are no expand all nodes and retract all nodes buttons. The second version of the spect view is a custom tree view which addresses all of these issues explicitly. In the Semplorer implementation, clicking on

²<http://protege.stanford.edu/>

any part of a node will expand or collapse the subtree. A search field with immediate feedback was implemented – every time a user enters a character all nodes matching the new string are expanded and highlighted. A collapse and expand all nodes button was introduced.

Another problem with traditional tree views is that only an inheritance relation is visualized. In the SempIorer version of the tree view this problem is addressed by adding all valid slinks for a category as child nodes of a category. A possible further development of this idea would be to make these nodes interactive by providing information about the other side of the relation when a user hovers over a slink. The view could also be made to navigate to the category on the opposite when the node receives a mouse click.

Whether these changes are an improvement over a traditional tree view is an open question as a second usability evaluation was beyond the resource frame for this thesis.

6.5 Visualizing Queries

Searching is an essential functionality in an information storage and retrieval system. In his famous intranet study³ Jakob Nielsen says that inadequate search capabilities were the single largest cause of reduced usability. Making accessible search interfaces is challenging because in most cases users are pre-occupied with finding the information they need. Learning how to translate their need into a language understood by the computer has a lower priority. Trying to elicit information from an unfamiliar system can quickly become a frustrating affair. It is like trying to get directions in a foreign country, the language of which you don't speak.

The users of search interfaces however do not behave like tourists in a foreign country. In the following the assumption is made that searching users behave like users of internet web search engines. It may be the case that usage of the SemDAV search interface evokes usage patterns different from the ones found on the internet. However, conducting a reliable study of SemDAV usage patterns is beyond the scope of this thesis. Existing studies will be used as an approximation.

Two of the most prominent web search engine studies have been considered as a baseline for user behavior [28, 61]. Search logs of the two search engines AltaVista and Excite were analyzed. Both studies have come to similar conclusions. For our purposes we will use only a few of their insights. First, the average number of search terms lies between two and three. Most users only

³<http://www.useit.com/alertbox/intranet-usability-1st-study.html>

use two terms in their queries. Second, users only look at the first page of results to their query. And third, boolean connectives were rarely used. Even when they were used, almost half of the time they were used incorrectly.

Although the design of the SemDAV search interfaces relies on all three of these assumptions, the most important observation is the one regarding boolean queries. On the abstraction level of the SemDAV protocol, the repository is queried with propositional boolean algebra. It has been shown time and again that boolean queries are very unintuitive to use for untrained users (cf. [8] section 10.5.1). This is probably the main reason for users shying away from using boolean operators in their internet queries.

The difficulties of boolean queries are manifold. First, the words AND and OR have subtly different meanings in natural languages and in logic. The phrase “dogs and cats” in natural languages refers to the union set of both at the same time, dogs *and* cats. The logical statement $dogs \wedge cats$ however refers to things that are both, dogs and cats. Similarly, the word *or* in natural languages has an exclusive meaning. Saying “I will rent a movie or go to the cinema” means that you will do either but not both. In logic the \wedge connective includes the possibilities that both conjunctives are true. A simple workaround to this ambiguity is to say “*any*” instead of “*or*” and “*all*” instead of “*and*” – referring to the fact that in the first case any of the propositions has to be true to satisfy the whole formula and in the second case all of the propositions have to be true for a formula to be satisfied. For example: “Any of: rain, wind, low temperature means that the weather is bad”. Or: “When something has all of: four legs, tail, canines, and barks then it is a dog”.

The second major difficulty imposed by boolean queries is the usage of parenthesis to nest formulas. Laws of associativity, commutativity and operator precedence are not understood by people without a formal background in logic. The search engine studies, which were used to formulate the searching behavior of users, have shown that the most commonly used connective was the logical “*and*” connective. This suggests that this connective is the most intuitive. The authors of the VQuery visual query language have come to similar conclusions [29].

In the literature several approaches have been proposed to facilitate creating boolean queries. The most fundamental concept with which this issue is tackled is direct manipulation as proposed by Schneiderman [58]. First, queries are constructed not by writing text but by pointing, clicking, dragging and dropping. Second, modifications of the query give immediate feedback – after every change of the query the results are updated. And third, flexible and forgiving facilities for manipulating the query are provided.

By putting the query language into a direct manipulation scaffold, a large

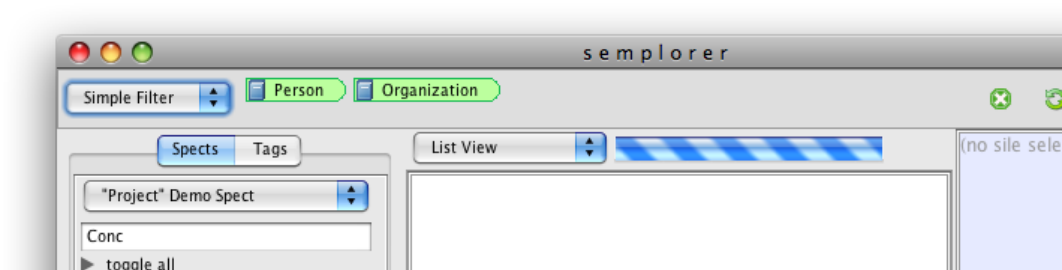


Figure 6.7: Simple filter.

class of errors is eliminated. By asking users to drop widgets into their queries, instead of typing formulas into a text box, syntax errors become impossible. Schneiderman further argues, that complex queries are created in an iterative refining process. Supporting this process with the user interface makes the learning curve of the query language flatter. Different versions of a query can be tried out in quick succession and the results of the queries can be browsed immediately.

Bending the tourism analogy from above even further – we are giving the tourist a magic box. With the help of the box she can immediately form complex sentences in the foreign language without making any grammatical mistakes. The box also tells her where the directions she received are leading – she will not have to walk them herself to find out.

6.5.1 Simple AND Filtering

It was mentioned above that the logical “*and*” is the most frequently used and easiest to understand connective. This insight paves the way for the simplest possible filter – a filter which allows combining terms only with a logical “*and*”. Adding terms to this filter reduces the number of results – each result has to satisfy a stricter and stricter constraint. The simplicity of this concept is its greatest strength but also its greatest weakness. Searching for an item in two sets becomes impossible. If you suspect that the item is in one of two sets, you have to search each subset separately.

Bob uses this interface when he is looking for a specific document and wants to iteratively narrow down his search criteria. Let’s say for example that his friend, Alice, has lent him her notes. Bob has categorized them in his SemDAV space and marked them as authored by Alice. Unfortunately, Alice lost her laptop and with it all lecture notes she has taken this semester. She asks Bob to give her copies of the notes she has given him for the theoretical computer science lecture. Bob first drags the category corresponding to the lecture into

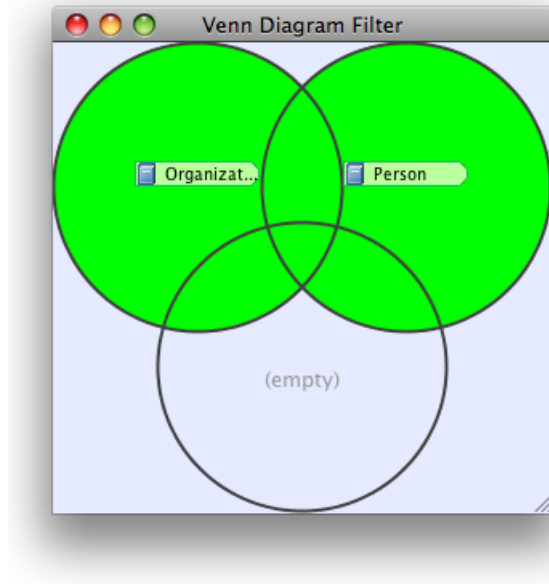


Figure 6.8: Venn diagram filter.

the filter shown in Figure 6.7. This gives returns all the siles stored in this category. To narrow down the results to only the ones authored by Alice he drags an author attribute into the filter and enters “Alice” in the value field of the attribute. The siles now listed in the sile view are exactly what Alice asked him for.

6.5.2 Venn Diagrams

The second interface for searching in a SemDAV repository allows using any combinations of connectives but is constrained insofar as it only allows combining only three terms. This constraint is motivated by the small average number of terms used in internet queries. The interface does not provide degrees of freedom which would allow forming invalid queries. The basic idea is treating terms as sets of entities which satisfy the term. Shown on the interface is a Venn diagram of all three sets so that all of them overlap (see also Figure 6.8). Users can then choose to include or exclude all possible subsets and intersections between the three sets. The decision was made to provide users with a premade template for three sets was made because the designers of the VQuery language [29] discovered that letting users arrange their own diagrams actually takes longer than formulating the equivalent boolean query in a textual form.

Bob usually uses this interface when he wants to narrow down his results using the inclusion/exclusion principle. For example, a part of Bob's notes in compiler development were scribbled in such a hurry that they are illegible even to him. He hopes that he can find this section in the notes of one of his friends. To do this, he drops the compiler development category into one of the set ovals of the Venn diagram. He drops the author attribute, with his name in it, into a second. Then he clicks on the subset of the attribute and the intersection of the attribute and the category to exclude all notes authored by him from the search results. Now he can browse all notes which were written by any of his friends in the site view.

The most obvious weakness of this method is that it only allows searches for a maximum of three terms. While Venn diagrams of more sets can be drawn, they become too unwieldy to have an advantage over formulating boolean queries [29]. Other possible weaknesses of this query method were revealed in an anecdotal interview with users without logical or set theoretical background. Having a template with three circles for three sets was perceived as an affordance to use exactly three query terms. Another issue was the inclusion or exclusion of set intersections. While it was easy to explain a scenario in which someone is searching for the intersection of two sets, the question of why one would want to exclude the intersection of two sets from the results could not be answered to the interviewees satisfaction. To make this query method a viable alternative to formulating boolean queries it is necessary to conduct controlled experiments and refine the method accordingly. Such research is however beyond the scope of this thesis.

6.5.3 Faceted Queries

The logical foundation of the third method lies in the formulation of faceted queries. The user interface is derived from the query reformulation interface in the AI-STARS information retrieval system [4]. An example of a faceted query is shown Figure 6.9.

Faceted queries divide a query into topics. Each topic is the union of the results of several keywords. The result of the faceted query is the intersection of all topics. An example for a faceted query is: $(osteoporosis \vee 'boneloss') \wedge (drugs \vee pharmaceuticals) \wedge (prevention \vee cure)$. A topic in this query is $(osteoporosis \vee 'boneloss')$. This topic describes all information which is about bone loss, osteoporosis or possibly both. The whole query yields results which are in all three topics.

Logically this corresponds to a boolean formula in conjunctive normal form (DNF). This is no restriction on the expressive power of queries, because any boolean formula has an equivalent formula in DNF. However, the conversion

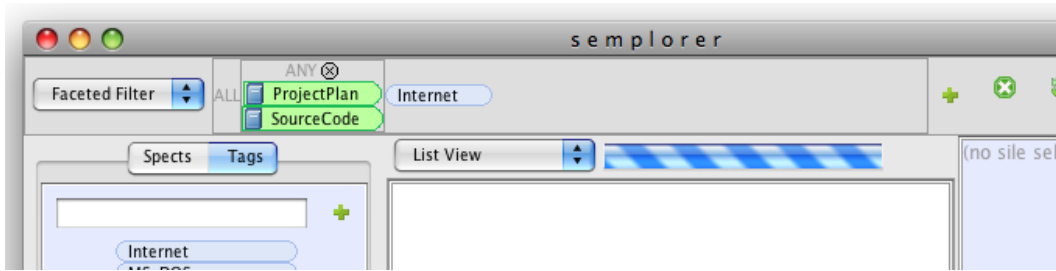


Figure 6.9: Faceted Filter.

to DNF may result in a formula which is exponentially larger than the formula the user entered. Consider for example the formula

$$(X_1 \wedge Y_1) \vee (X_2 \wedge Y_2) \vee \dots \vee (X_n \wedge Y_n) \quad (6.1)$$

the equivalent formula in CNF has 2^n clauses where n is the number of clauses in the original formula:

$$(X_1 \vee \dots \vee X_{n-1} \vee X_n) \wedge (X_1 \vee \dots \vee X_{n-1} \vee Y_n) \wedge \dots \wedge (Y_1 \vee \dots \vee Y_{n-1} \vee Y_n) \quad (6.2)$$

Faceted queries are simple enough to be explained to someone without prior knowledge of boolean algebra. This kind of queries does not make use of properties which have been found to be confusing to novices. The most prominent example of a confusing property is the usage of parenthesis to nest logical formulas. An overall upside of this method is that it does not restrict the expressive power of boolean algebra. The simplicity of the method should however not be given too much weight. Anecdotal interviews have shown that of the three implemented methods faceted queries are the most difficult to understand by lay people. The other downside of faceted queries is that they are equivalent in expressiveness to other formulas only in theory. Some queries which have terse representations may become unwieldy when reformulated into DNF.

This method was developed mainly to get a feeling for the tradeoff between expressive power and usability. As such there are no convincing scenarios in which Bob would use this interface. The interface is geared towards professional information workers but is not deployable without further investigations.

The least attention was paid to the logical negation in queries. All through the cited literature the negation operator was found to be seldomly used. The authors of the AI-STARS interface [4] even omitted negation altogether from their otherwise sophisticated user interface. In the Semporer interface entities are negated by rightclicking on them and choosing “*exclude*” from the context

menu. Negated terms are made positive, again by use of the same context menu.

6.6 Introduction of New Vocabulary

A pragmatic solution for applications with a universal purpose is to translate as much terminology as possible into a language understood by the majority of the intended audience. In cases where this is not possible, there are two options. The first option is to introduce new terms which are consistent with the rest of the application, in which case users will have to learn only the new terms: having to learn $n - k$ new words (where n is the total number of words and k is the number of known words) is still an improvement over having to learn n new words. The second option is to omit functionality which requires the understanding of foreign terms. As often is the case, any solution for this problem will be a tradeoff between the expressive power of the application on the one hand, and the steepness of the learning curve for new users on the other hand.

The choice of terminology in the SemDAV user interface follows this rationale. The application uses terms that are used in common language, such as “*category*” or “*filter*”. In some cases it was safe to assume that users were familiar with very similar concepts to the ones we employed, hence we tried to choose words that are phonetically similar to the concepts already known to users but still accentuate the difference. The word “*sile*”, for example, is very similar to “*file*”, and “*slink*” is very similar to a “*link*” on the web, but nevertheless, slightly different. Unfortunately there are still more concepts which are unknown to the majority of the intended user population, like for example the word “*ontology*”. In such cases we have deliberately used neologisms in order to avoid ambiguities. This naming scheme has however not been formally evaluated so there is no evidence that it makes the slope of the learning curve flatter.

Chapter 7

Software Engineering Aspects

The ultimate goal for the Semporer is to provide a rapid prototyping platform for experimenting with semantic user interface concepts. This goal imposes several constraints on the software architecture of the application. The framework needs to implement frequently needed functionality so that it actually saves effort. Optimally the framework provides a functional application which can be easily modified. Components of the application need to be easily replaceable and removable without impact on the remaining application. To satisfy these constraints the framework has to provide well defined and orthogonal interfaces between modules. The modules themselves need to be as loosely coupled as possible. The following chapter describes the different components of the Semporer application emphasizing on how they satisfy the constraints of having clean interfaces and being loosely coupled.

7.1 Semporer Architecture

The architectural layout of the Semporer is a reflection of its visual appearance. For example, the sile pane is a single module which is responsible for managing all the sile viewers which can be selected in the dropdown menu. Special attention has been paid that more general components are less coupled. The panes themselves do not communicate directly with any other GUI components. Only two types of necessary communication were identified: communication required for drag and drop gestures and the necessity of broadcasting the currently selected siles. These two types of interaction are implemented in two separate modules. A third module which communicates with GUI component is a custom event framework which notifies GUI components about interaction with the SemDAV repository. Figure 7.1 shows a birdeye view of the Semporer architecture. Note that the figure only gives a summary of the general structure of the application, many details were omitted.

Taking advantage of the modularity of the Semporer works best when new components stay consistent with the remaining application. For example, a central concept for interacting with SemDAV entities are drag and drop gestures. New components which adhere to this interaction “language” by reusing the SemDAV entity widgets provided by the framework do not need to implement any additional functionality to integrate with the Semporer. This constraint is in practice a very soft one because the imposed interaction

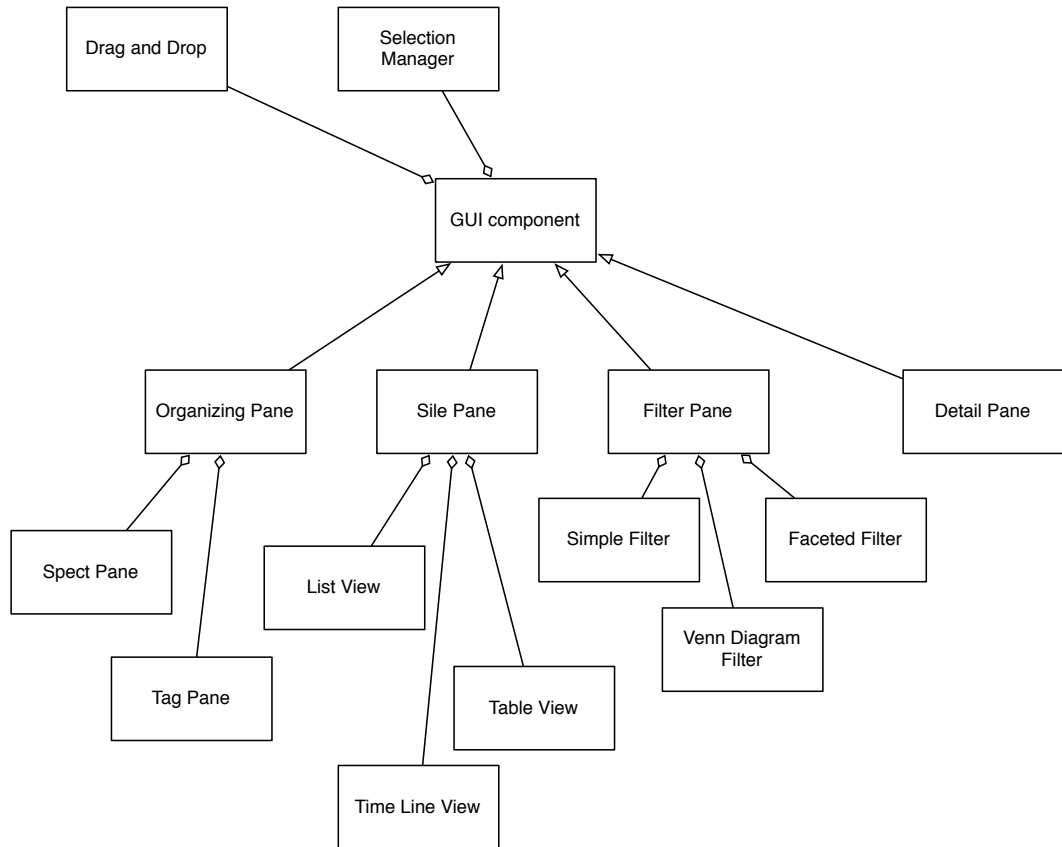


Figure 7.1: Birdeye View of the Sexplorer Architecture.

language is composed of very few gestures and visualization concepts. The average user can be expected to be already familiar with these concepts and gestures from other applications.

7.1.1 Panes

As mentioned above, the visual appearance of the Semporer is a reflection of the source code organization. Each of the visible panes is located in its own package. There are four types of panes: organizational, visualizational, informational and filter panes. In the following each panel type will be described in more detail.

Organizational Panes

The organization package contains the panes which are used for organizing siles. Currently these are the spect pane and the tag pane. The spect pane provides a user interface for navigating spect. Because the navigational nodes are simply SemDAV entities, they can be dragged and dropped onto a sile to categorize the sile. The category widgets are displayed in a tree like structure. Because the default Swing¹ tree component does not make it possible to use custom widgets for nodes and leafs, a custom tree component was developed. This component, called the SpectBrowser, resides in its own package. It is kept generic so that it does not rely on other Semporer specific widgets. If the navigational behavior of the spect pane needs to be modified it suffices to change the SpectBrowser component.

Visualization Panes

The central package of the semporer – at least in a spatial sense – is the visualization package. It implements visualization methods for siles. Similarly to the filtering package, there are three different visualization methods with varying complexity. The list view is a simple sorted list of all siles. While the usefulness of this view may be limited, it is a good starting point for developing new visualizations. The other two visualizations, the timeline view and the table view, are quite complex. The table view is a Swing centric visualization, it utilizes components which are implemented as a part of the Swing framework. The timeline view on the other hand is an approach which implements interaction and visualization concepts from scratch. Both were implemented for usability experimentation and analysis. While modifying them is certainly not discouraged, their use as boiler plate code for new interaction methods may be

¹java.sun.com/docs/books/tutorial/uiswing/

limited because of their relative complexity. While not a sile visualization in the strict sense, the sile pocket is also contained in this package. The reason is that this pane is structurally very similar to the simple list view.

Informational Panes

The only pane in the *informational* package at this time is the detail pane. It may be the only component located in this package but it is sufficiently complex and orthogonal to the other components to warrant having its own package. The main reason for the complexity of the detail pane are its collapsible regions. Since Swing provides no such functionality, they had to be developed from scratch. These regions are implemented with the help of the so called WidgetAdder. This class provides a pane to which widgets can be added. It further provides a header region which is basically a simple Swing JLabel extended with a triangle which gives feedback about extending and collapsing the region. It is interesting to note that the widgets added to this component are layed out with the help of the custom built wrap layout manager. We will however cover custom layout managers in the Sexplorer in more detail later.

Filter Panes

Another important package is the filter package. This package contains the filtering user interfaces. In particular these are the simple filter, the faceted filter and the Venn diagram filter. When writing new filter user interfaces it is instructive to study the source of the simple filter. It is not only the most simple and limited user interface but also has the most simple source code. It consists of only one class. It is a good example of how to write new filters. The faceted filter is a little more complex, it can be thought of as simple filters nested in a simple filter. The Venn diagram filter is not so much an instructive demonstration as it is an example of how far the SemDAV filtering interface can be pushed.

7.1.2 Widgets

There is a group of components which are not designed for easy replacing but for easy reusing. These are the SemDAV widgets, they are a group of widgets, conceptually comparable to a complex version of Swing's JLabels, which represent the central SemDAV concepts. There is a widget representing a category, one for tags, one for attributes, one for slinks and one for siles. All direct manipulation techniques are implemented inside the SemDAV widgets. The

direct manipulation functionality is implemented in layers. This corresponds to a factory pattern putting together decorators. For example, if you want to place a widget somewhere which can be removed by pressing its “x” button, you invoke the factory method which generates a removable version of the desired widget. The factory in turn takes care that the removable decorator is placed onto the widget. All decorators inherit from a common class, the `SemdavDecorator`. The main purpose of this class is to implement drag and drop feedback functionality. For example when you drag a widget over another widget which is a valid drop target, the `SemdavDecorator` is responsible for highlighting the target. Several different decorators have been implemented:

border decorator is responsible for the graphical representation of the widgets.

removable decorator as mentioned above, this decorator adds a button with which a widget can be removed. The actual removing procedure needs to be reimplemented depending on the context of the widget.

editable decorator displays an edit field over the widget allowing users to change its value.

selectable decorator is responsible for visualizing selections of multiple widgets.

tooltip decorator displays tooltips when a user hesitates over a widget.

negator decorator this last layer is needed only for the filter panes, it renders the widget as logically negated.

browsable decorator is a decorator which makes a component compatible with the `SpectBrowser` component.

Note that although the common definition of the decorator pattern implies that the order in which decorators are put on top of each other should not matter, this is not true for the semplorer decorators. Developers should use the factory to build new widgets. If a widget is required which is not yet implemented in the factory, a deviation from the order of decorators seen in other factory methods may cause subtle problems.

7.1.3 Infrastructure

Let us now take a closer look at the packages which are not implementing parts of the graphical user interface. There are two of these, an event dispatching framework and the selection manager.

Communication

SemDAV is a synchronous protocol, this means that a client can perform only one action at a time. However, because SemDAV is based on XML-RPC over HTTP, it is technically cheap to send multiple requests to the SemDAV repository in parallel. On both sides of the communication some scheduling and serialization has to take place to keep track of the order of requests and possible conflicts. On the client side this is accomplished within a separate module, which is described in more detail in the following section. This module wraps around the synchronous XML-RPC stub and provides non blocking SemDAV operations. Because the methods are non blocking, any results returned by the calls are not returned in the context of the caller. Instead, upon return of a SemDAV operation the communication framework triggers an event, attaching any results to it, which is then propagated to all subscribers. If a caller is interested in the results of a call it needs to subscribe for the event which will deliver the resulting objects.

An event framework is required for this mechanism to work. One option would be to use the AWT event framework². This framework however is very inflexible. In spite of the introduction of a generic type system in recent Java versions, it is necessary to write a firing method for every event. This is not only cumbersome but also error prone because the event dispatching code is changed for every newly introduced event type. To overcome this limitation a new event dispatching framework was implemented. The main difference between the Semplorer event framework and the AWT framework is that the Semplorer framework does not rely on Java types for dispatching events. Instead, all events are of the same Java type. They are told apart with a string identification member of the event. This amounts to replacing the dispatching mechanism. Instead of using the virtual machine to go through all methods in a dispatching object and decide which to call, the Semplorer makes the dispatching code explicit. The Semplorer keeps a list of all subscribers and every time an event is fired, the dispatcher iterates through this list, dispatching the event to subscribers for the event. An additional upside of using a custom event dispatching framework is that it is implemented in a thread safe manner.

Selection Management

The second central component which is used by all user interface parts is the selection manager. The selection manager is responsible for keeping track of which files are selected in the file view pane and providing this information to other components. It is interesting to note, that the selection mechanisms

²<http://java.sun.com/products/jdk/awt/>

are handled largely by the sile widgets. This means that only minimal effort is required to allow selecting multiple siles in a new visualization component.

Layout Management

The Swing layout managers were unfit for several of the mechanisms required in the *semplorer*. In these cases new managers had to be developed. The main reason for designing new layout managers was that the *Semplorer* largely relies on automating wrapping of lines which do not contain text instead of widgets. Wrapping widgets goes against the Swing layouting principles so it can only be implemented with some workarounds. To wrap correctly the layout manager needs to know how much space it has available. It can then fill up a line and put a widget that does not fit on the same line on the next line. In Swing however the layout process is a bottom up process: a component which lays itself out first asks all its children to layout themselves and tell how much space they require. A child which wants to do wrap layouting however expects the parent to tell it how much space is available.

In the *Semplorer* this chicken-egg problem is solved as follows: the child component tells its parent that it requires zero space. When the parent has finished laying out it fires a Swing event. The wrap layout manager listens for this event and as soon as it arrives, it initiates the real layout process for the child because now the parent has an assigned width. Similarly an event triggers layouting anew when the parent is resized by user interaction.

Miscellaneous Packages

The remaining two packages are not essential parts of the *semplorer*, they merely support its look and feel. The appearance package controls things like colors, fonts and icons for different widgets. The general principle is that a component asks classes in this package for appearance information, passing itself as an argument. The appearance classes return the requested information as configured for the type of the caller.

Finally, the *utils* package contains classes which are shared between more than one component and classes which do not fit anywhere else. A widely shared class is the rotating triangle which is used to give feedback on the collapsed and expanded status of widgets. A class that is in the *utils* package because it does not fit anywhere else is the *SileOpener*. When a user double clicks a sile it is opened in the application responsible for its mime type. The *SileOpener* controls the downloading of the sile content and opening the right application.

Figure 7.2 provides a summary of the Semporer packages and their containment.

7.2 Concurrent Communication Module Architecture

The concurrent communication module is strictly speaking not a part of the semporer. Because of the relatively tight coupling between the two modules it is still justified to review it in some detail. The main purpose of the concurrent module is to expose non blocking SemDAV operations on top of the sequential SemDAV protocol. This task is less complex than it sounds. The SemDAV protocol is sequential but it is based on XML-RPC which is transferred over HTTP. This implies that the server side can be based on servlets. The advantage of using servlets is that a new context is spawned for every call to the repository for free. Although behind the servlets the calls are still queued, the performance gain is still significant since often the greatest bottleneck is the transmission and not the processing of the calls.

For example, when populating the sile pane, a few default attributes are requested for every sile which is about to be displayed. One of the more obvious attributes which are needed is the sile name. Another is the MIME type of the sile which is used to attach a meaningful icon to the sile. One of the goals in the design of the SemDAV protocol was to keep it simple so that it can be reimplemented with minimal effort. Because of this simplicity there is no API support for fetching several groups of attributes belonging to different siles. Instead, this is done by issuing one method call for every sile. Because these calls are cheap and large in number, parallelizing them brings a large performance benefit.

Of course, for applications in which responsiveness is not an issue, the concurrent module does not need to be utilized, it can be entirely omitted and the client can use the XML-RPC stub directly.

7.2.1 Scheduling

The client side counterpart is slightly more complex mainly because the XML-RPC framework is not thread safe. Executing XML-RPC calls concurrently on the same point results in unpredictable behavior. The core of the concurrent module is a scheduling thread which runs decoupled from the rest of the application. SemDAV operations are wrapped in command objects which are put in a queue. The scheduling thread takes operations from this queue and executes the corresponding XML-RPC calls. The scheduler is responsible for serializing calls in such a way that undesired side effects are kept at a mini-

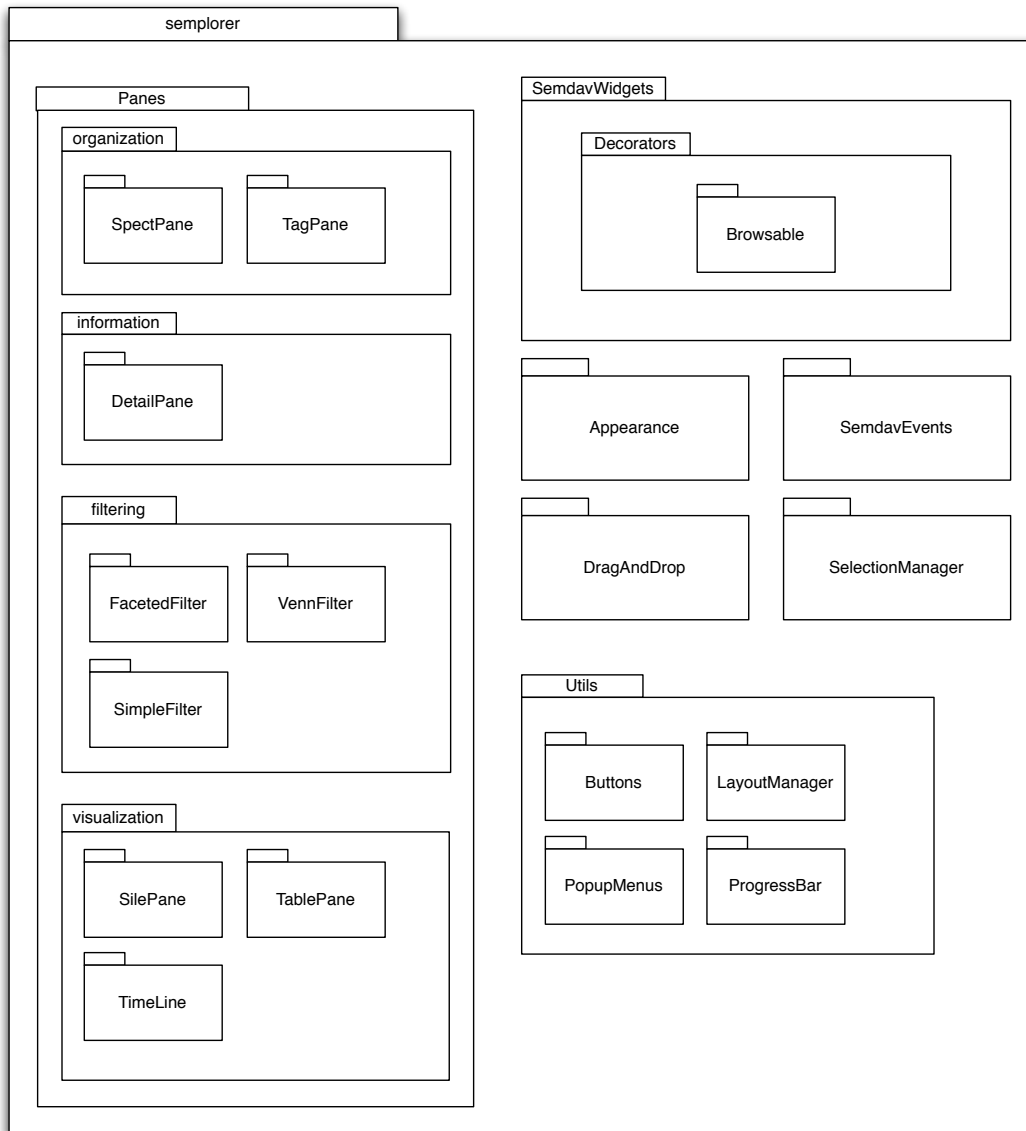


Figure 7.2: Package containment diagram for the Semplorer.

mum. Executing two write operations on the same resource at the same time for example should not be possible. The consistency mechanism in the current scheduler is fairly simple, it was deemed that implementing sophisticated consistency enforcing is out of the scope of a user interface prototype. Currently the scheduler executes read operations in parallel, but only one write operation at a time. In preliminary tests the case that a resource is written and read at the same time did not occur. However, an eventual next iteration prototype will have to focus on concurrency issues.

The scheduler has a few more responsibilities next to coordinating SemDAV operations. One of these responsibilities is keeping track of the undo and redo subsystem. Every action which is executed is placed on the undo stack. When an undo operation is triggered, a single action is popped from the undo stack, its undo method is executed and the action is placed on the redo stack. When doing a redo, the command is executed normally, using the execute method. A possible improvement in future versions would be to only put actions on the stacks if they completed successfully, after all the semantics of undoing a failed action are not well defined.

Another task of the scheduler is keeping the semplorer up to date on the progress of the actions currently in the queue. This is kept rather simple: a counter is increased for every new action and decreased for every completed action, every time the counter changes, the progress handler is notified. The client decides how to visualize how many actions are remaining. Currently the client visualization only differentiates between busy and not busy states. This is visualized by a progress bar above the file pane. In future prototypes it would be easy to display progress percentages with the bar rather than just busy and ready states.

7.2.2 Caching

Profiling measurements showed that the semplorer frequently requests the same information several times. The conclusion drawn was that performance and thus responsiveness of the user interface can be achieved by implementing a caching mechanism. The caching was implemented in the concurrent module as the main purpose of this module is performance optimization. The mechanism does not rely on a framework, it is highly domain specific. It was found that a custom implementation provides the best control over caching strategy and granularity at a minimal implementation effort. Further, some quirks of the SemDAV implementation make it necessary to have tight control on the order in which operations are carried out. For example, when populating the cache, three attributes are fetched: the label, the creation date of the resource and the content type. The three attributes are passed as an array

to the method together with an array of siles and the API returns these attributes for the requested siles. However, some siles do not have a creation date. A typical example of such siles are siles which are hosted on a remote machine – in such cases there is no way to infer a creation date. If the creation date is first in the attribute array, then remote siles will be omitted from the result. The reason is that the query engine tries to find a creation date, sees that there is none, and decides that it is not worth it to consider the sile any further. Further implementation details on this peculiar behavior are outside of the scope of this thesis.

The mechanism itself is fairly conservative. It is necessary to keep in mind, that the central entity type in SemDAV is the sile, while every other entity type is used to describe siles. In this paradigm it makes best sense to cache entities related to specific siles. If information is requested about a sile, the cache can answer.

When requesting the siles matching the current filter, the cache is populated with all the entities necessary to render a sile widget, it is assumed that the application will need these soon. The specification of the SemDAV protocol makes it possible to request several types of annotations for several siles in one go. Requesting the annotations for all siles which are soon going to be drawn at once and caching them is a lot faster than requesting the same annotations separately for each sile. More than these few entities are rarely needed. More specifically, more annotations are requested only when the detail pane is filled. The detail pane however is filled not only with more but with all annotations for a specific sile. So it is safe to assume that if a cache request misses, all annotations for the specific siles can be fetched into the cache. Subsequent cache requests, presumably performed by the detail pane, will all hit cached entities.

Clearing the cache is similarly defensive. When a new sile search is initiated the cache is cleared entirely and populated from scratch. A scenario in which the cache is only partially cleared is when a write operation on a sile is performed. Again, it has to be pointed out that the main purpose of implementing the caching mechanism was to make the user interface responsive. To use a SemDAV client in a multi user environment further work is necessary to produce a consistent and performant cache.

7.2.3 Lifecycle of a Method-Call

Figure 7.3 shows a slightly abstracted overview of the execution of a SemDAV operation. It demonstrates the internal processing of the concurrent module. In the first step, the Semplorer requests the execution of some SemDAV operation. The request is sent to the SemdavManager. The SemdavManager

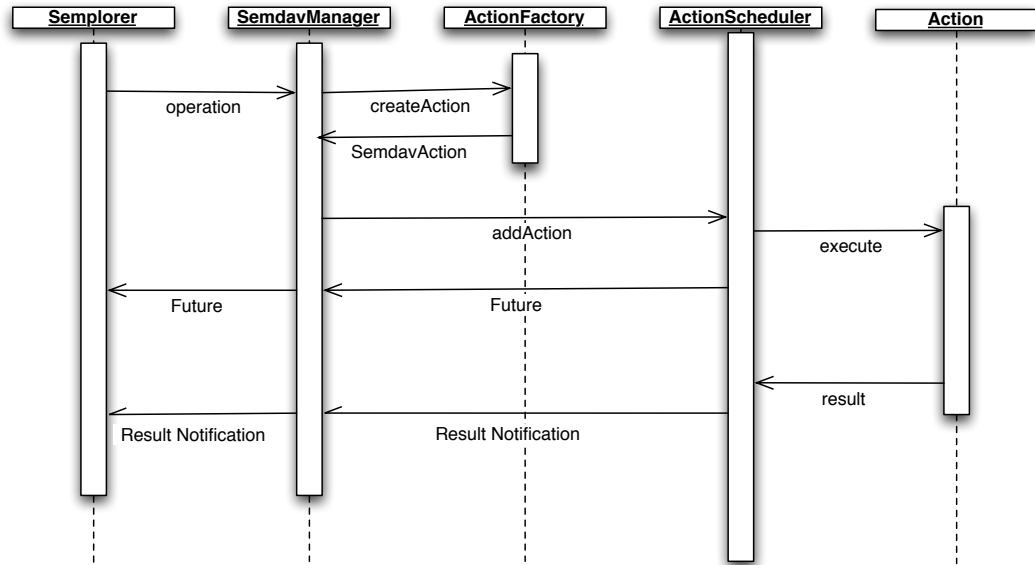


Figure 7.3: Sequence diagram of a concurrent SemDAV operation.

is a facade class to the concurrent module, the Semplore does not need to interact with any other class in this module. The Manager then asks the class `SemdavActionFactory` to construct a command object corresponding to the requested operation. As pointed out by Gamma et al. [22], storing operations using the command pattern has several advantages. Two of the most prolific are: first, it is easy to add undo/redo functionality when a command interpreter is present, and second, we can define for each operation separately how it influences the cache in a maintainable manner. After the command object has been constructed it is returned to the Manager who in turn adds the command to the execution queue of the `ActionScheduler`. The queue in the scheduler is thread safe, it is continuously processed by the scheduling thread which is defined entirely in the `ActionScheduler` class. In response to the queuing the scheduler returns a `Future` object. The `Future` class is defined in the Java concurrency framework³. The `Future` object provides information about the state of a command which is currently in the execution queue. It is possible to poll the `Future` object for results when synchronous communication is desired, otherwise this object can be safely ignored. When the command has completed execution, the scheduler triggers a callback in the Manager, the Manager then notifies the Semplore that an action has finished execution. The semplore then decides which events to trigger for the finished operation in order to let user interface component know that they may need to update

³<http://java.sun.com/j2se/1.5.0/docs/guide/concurrency/>

themselves.

This depicting is, as mentioned above, slightly abstracted. The simplification lies in how the manager relies responses back to the Semporer. To receive responses the Semporer needs to register a so called condition listener with a condition handler which is located in the manager. Every time an operation completes execution, the scheduler notifies the condition handler, which in turn notifies all condition listeners. Information about the action's status is relayed by simply passing around the command object. The command object contains additional information, like a completion status, a textual description of the action and others. Using this information it is easily possible to decide how to react in the user interface. For example, if the completion status indicates a failure, the user interface can display an error message saying which command failed.

Chapter 8

Experimental Results

In the SemDAV project the resources for usability evaluation are limited. Acquiring an external usability expert was not an option, so cheap evaluation methods that could be performed by inexperienced evaluators were needed. There are not many methods that fulfill these constraints; we considered *heuristic evaluation* [37–39], *think aloud evaluation* [39] and *paper prototyping* [62]. Nielsen [39] advises to apply at least two methods for acceptable accuracy. We have chosen to combine a heuristic evaluation and a think aloud evaluation because of their simplified administration in comparison to that of a paper prototyping evaluation. In the following we outline the rationale behind our selection.

Because the team involved with the development of the SemDAV application had no prior experience with a heuristical evaluation, the obvious first step was to introduce the method. The evaluators were acquainted to the guidelines and their interpretation in a tutorial session, and the interpretations encompassed by every guideline were elaborated. The understanding of each guideline was ensured by the demonstration of a violation in commonly available software. Finally, a heuristic evaluation exercise [36] was discussed. The questions that were asked during this discussion anticipated the biggest difficulties the evaluators had during the evaluation. One question was how much time to spend on the evaluation, another question was how many problems the evaluators were expected to find. The third big problem was the format of the evaluation protocol. Care was taken to leave the answers to these questions as open as possible in order to ensure that the evaluators were not influenced by constraints.

In retrospect this may not have been an optimal decision. We recommend to not only present and describe an example of an evaluation, but to actually perform a simple evaluation in the course of the introductory tutorial. While this makes the preparation and administration of the tutorial more time consuming, the extra effort has manifold returns in terms of a common understanding of the method that is shared by all participants.

The most critical part of a heuristic evaluation is the necessary familiarization of the assessment team with the evaluation method. The guidelines by themselves are worded in a very general manner which is prone to misunderstanding. For example, consider Nielsen's first guideline:

Simple and natural dialogs: *Dialogs should not contain information*

that is irrelevant or rarely needed. Every extra unit of information in a dialog competes with the relevant units of information and diminishes their relative visibility. [39]

We discovered that several of our evaluators misunderstood what was meant by the word “*dialog*”. Nielsen’s intended meaning of this term is the way the user interface communicates with the user to help her achieve a goal. Some evaluators however assumed that this guideline only applies to dialog boxes. A good way to alleviate problems like this is a short discussion of possible interpretations in the tutorial session.

Even with a good instruction, detecting conflicts with the heuristics is a difficult task. Studies performed by Nielsen and Molich [37] have shown that, depending on their experience and knowledge of user interfaces, single evaluators discover about 20% of all usability problems. Thus the method only yields acceptable results if several persons independently perform the evaluation. The overlap of discovered problems between individual evaluations is very low: the cumulative results of 3 to 5 evaluators account for about 50 to 80 percent of all evaluation problems [37]. Still, the quality of the results highly depends on previous experience with usability inspection methods. It is unclear however whether this estimation also holds for more complex applications that require a longer time to familiarize.

In our evaluation, we found 50 issues with 5 overlaps. This amounts to 45 issues among 4 evaluators. Comparing it to heuristic evaluations of other applications, there is a reason to believe that the real number of issues is larger. A rough interpolation on benchmark results of the method [37] make it safe to assume that we have found between 50 and 75 percent of all usability problems. Nevertheless, the evaluation yielded considerable results considering the low volume of invested effort.

Chapter 9

Conclusion and Future Work

The thesis at hand identifies six challenges which need to be addressed by developers of semantic information systems. These challenges are: how to navigate large data sets, how to design for diverse users, how to visualize ontologies for information retrieval rather than ontology engineering, how to make the formulation of boolean queries intuitive, how to model the semantics of resource names and how to introduce unfamiliar terminology. Related projects were reviewed with respect to these challenges. The outcome was that none of the related projects address all of these issues.

Three of these challenges were addressed in the Semporer prototype. The visualization of large data sets is tackled with a timeline centric navigation paradigm. Several approaches were considered for the visualization of ontologies. The final prototype implements a specialized tree view as proposed by Katifori et al. [2]. For the formulation of boolean queries the thesis proposes three visual methods: a filter based on the usage of only a logical *and* connective, a filter using interactive Venn diagrams to visualize boolean formulas, and a filter using faceted queries inspired by the user interface in the AI-STARS project [4].

A byproduct of the prototypical implementation is a framework for rapid development of semantic user interfaces. The software architecture of the Semporer was designed to optimize genericity. Widgets can be easily reused in new visualizations so that developers can concentrate on interaction concepts rather than appearance details.

A preliminary usability evaluation of the Semporer was conducted. The evaluation has shown that the Semporer is ready for deployment in the field. While the findings of the evaluation were considered in subsequent prototype iterations, not all shortcomings could be ironed out. Prior to a deployment of the Semporer further usability evaluations need to be administered.

In followup work the highest priority would be placed on addressing the remaining three challenges: designing for diverse users, semantics of resource names, and introduction of new vocabulary.

A topic which could not be addressed in this thesis because of limited resources was the application of generalized fisheye views as described by Furnas [21]. While he designed them for ASCII terminals, they can be applied to graphical user interfaces as well. For example, it can be investigated whether fisheye-like time intervals in the timeline view bring any improvement. In such

a timeline the intervals which are nearest to the center of zoom would be widest and the intervals furthest away would be longest. Another topic which was out of the scope of the thesis was the implementation of a site visualization method using animated dynamic graphs with radial layout as described by Yee et al. [67] combined with fisheye information abstraction.

One of the promising future directions of the Semplorer user interface is the investigation of its applicability in mobile devices. Open questions are: how the drag and drop paradigm translates to mobile devices, which site views can be adapted to the limited screen space of a mobile device, how to cache information for use without a network connection.

To treat the questions of mobile semantic applications a multi user collaboration model has to be devised. While this is not so much a scientific as it is a technical challenge, it's convincing solution is essential to the further development of the SemDAV project.

Bibliography

- [1] Nitin Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. A Five-Year Study of File-System Metadata. In *Proceedings of the 5th Conference on File and Storage Technologies (FAST '07)*, San Jose, CA, 2007.
- [2] Katifori Akrivi, Torou Elena, Halatsis Constantin, Lepouras Georgios, and Vassilakis Costas. A comparative study of four ontology visualization techniques in protege: Experiment setup and preliminary results. In *IV '06: Proceedings of the conference on Information Visualization*, pages 417–423, Washington, DC, USA, 2006. IEEE Computer Society.
- [3] Harith Alani. Tgviztab: An ontology visualisation extension for protege. In *Knowledge Capture 03 — Workshop on Visualizing Information in Knowledge Engineering*, 2003.
- [4] P. G. Anick, J. D. Brennan, R. A. Flynn, D. R. Hanssen, B. Alvey, and J. M. Robbins. A direct manipulation interface for boolean information retrieval via natural language query. In *SIGIR '90: Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 135–150, New York, NY, USA, 1990. ACM.
- [5] Margaret anne Storey, Mark Musen John Silva, Neil Ernst, Ray Ferguson, and Natasha Noy. Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition. In *Protégé. Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001)*, 2001.
- [6] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press, April 2004.
- [7] Apple. Apple human interface guidelines, 2005.
- [8] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, May 1999.
- [9] Tim Berners-Lee. *Weaving the Web*. Texere Publishing Ltd., November 1999.
- [10] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, pages 34–43, 2001.

- [11] Stephan Bloehdorn, Olaf Görlitz, Simon Schenk, and Max Völkel. Tagfs - tag semantics for hierarchical file systems. In *Proceedings of the 6th International Conference on Knowledge Management*, 2006.
- [12] M. Ted Boren and Judith Ramey. Thinking Aloud: Reconciling Theory and Practice. *IEEE Transactions on Professional Communication*, 43:261–278, 2000.
- [13] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, August 1995.
- [14] Hinrich Schütze Christopher D. Manning, Prabhakar Raghavan. *Introduction to Information Retrieval*. Cambridge, 2008.
- [15] Alan Cooper, Robert Reimann, and Dave Cronin. *About face 3: the essentials of interaction design*. John Wiley & Sons, Inc., New York, NY, USA, 2007.
- [16] Edsger W. Dijkstra. The threats to computing science. circulated privately, November 1984.
- [17] Laura Downey. Designing Annotation Mechanisms with Users in Mind: A Paper Prototyping Case Study from the Scientific Environment for Ecological Knowledge (SEEK) Project. In *Proceedings of the Semantic Web Personalization Workshop at the ESWC 2006*, 2006.
- [18] Susan T. Dumais and Thomas K. Landauer. Using examples to describe categories. In *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 112–115, New York, NY, USA, 1983. ACM.
- [19] A. Edmunds and A. Morris. The problem of information overload in business organisations: a review of the literature. *International Journal of Information Management*, 20(1):17–28, 2000.
- [20] Eric Freeman and David Gelernter. *Beyond the Desktop Metaphor — Designing Integrated Digital Work Environments*, chapter Beyond Lifestreams: The Inevitable Demise of the Desktop Metaphor, pages 19–48. MIT Press, 2007.
- [21] G. W. Furnas. Generalized fisheye views. In *CHI '86: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23, New York, NY, USA, 1986. ACM Press.

- [22] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, January 1995.
- [23] Jorge Gracia, Vanessa Lopez, Mathieu d'Aquin, Marta Sabou, Enrico Motta, and Eduardo Mena. Solving Semantic Ambiguity to Improve Semantic Web based Ontology Matching. In *Proc. of the 2nd Ontology Matching Workshop (OM'07) at the 6th International Semantic Web Conference (ISWC 2007)*, November 2007.
- [24] Haystack Group. Haystack website. <http://haystack.lcs.mit.edu/>, February 2008. outdated reference.
- [25] Tudor Groza, Siegfried Handschuh, Knud Möller, Gunnar Grimnes, Leo Sauermann, Enrico Minack, Mehdi Jazayeri, Cédric Mesnage, Gerald Reif, and Rósa Gudjónsdóttir. The nepomuk project - on the way to the social semantic desktop. In *Proceedings of I-Semantics' 07*, pages 201–211, 2007.
- [26] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, October 1999.
- [27] David Huynh, David R. Karger, and Dennis Quan. Haystack: A platform for creating, organizing and visualizing information using rdf. In *Semantic Web Workshop, 2002*.
- [28] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32(1):5–17, 1998.
- [29] Steve Jones, Shona McInnes, and Mark S. Staveley. A graphical user interface for boolean query specification. *International Journal on Digital Libraries*, 2:207–223, 1999.
- [30] David R. Karger and Dennis Quan. Haystack: a user interface for creating, browsing, and organizing arbitrary semistructured information. In *CHI '04 extended abstracts on Human factors in computing systems*, pages 777–778, 2004.
- [31] R. Kullberg. Dynamic timelines: Visualizing historical information in three dimensions, 1995.
- [32] John Lyons. *Semantics*, volume 5 of *Handbücher zur Sprach- und Kommunikationswissenschaft*, chapter General Foundations, pages 1–24. Walter de Gruyter Berlin New York, 1991.

- [33] Thomas W. Malone. How do people organize their desks?: Implications for the design of office information systems. *ACM Trans. Inf. Syst.*, 1(1):99–112, 1983.
- [34] Catherine C. Marshall and III Frank M. Shipman. Spatial hypertext and the practice of information triage. In *HYPertext '97: Proceedings of the eighth ACM conference on Hypertext*, pages 124–133, New York, NY, USA, 1997. ACM.
- [35] Danish Nadeem and Leo Sauermann. From philosophy and mental-models to semantic desktop research: Theoretical overview. In Tassilo Pellegrini and Sebastian Schaffert, editors, *Proceedings of I-Semantics' 07*, pages pp. 211–220. JUCS, 2007.
- [36] Jakob Nielsen. Improving a human-computer dialogue. *Communications of the ACM*, 33:338–348, 1990.
- [37] Jakob Nielsen. Finding Usability Problems Through Heuristic Evaluation. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 373–380, New York, NY, USA, 1992. ACM Press.
- [38] Jakob Nielsen. The usability engineering life cycle. *Computer*, 25:12–22, 1992.
- [39] Jakob Nielsen. *Usability Engineering*. Academic Press, 1993.
- [40] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, pages 249–256. ACM, ACM New York, NY, USA, 1990.
- [41] School of Electronics and South Hampton Computer Science. mspace. <http://mspace.fm/>, 10 2008.
- [42] C. K. Ogden and I. A. Richards. *The Meaning of Meaning*. Harcourt, 1989.
- [43] T. I. M. O'Reilly. What is web 2.0: Design patterns and business models for the next generation of software. *Social Science Research Network Working Paper Series*, page 17, 2007.
- [44] Venkateswaran S Prashanth Mohan, Raghuraman and Arul Siromoney. Semantic file retrieval in file systems using virtual directories. Poster at HiPC 2006, December 2006.

- [45] Microsoft Press. *Microsoft Windows User Experience: Official Guidelines For User Interface Developers And Designers*. Microsoft, 1999.
- [46] D. Quan, D. Huynh, and D. Karger. Haystack: A platform for authoring end user semantic web applications, 2003.
- [47] D. Quan and D. Karger. How to make a semantic web browser, 2004.
- [48] Dennis Quan, David Huynh, and David R. Karger. Haystack: A Platform for Authoring End User Semantic Web Applications. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*. Springer, 2003.
- [49] Dennis Quan, David Huynh, David R. Karger, and Robert Miller. User interface continuations. In *Proceedings of UIST 2003*, pages 145–148, 2003.
- [50] Jun Rekimoto. Time-machine computing: a time-centric approach for the information environment. In *UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 45–54, New York, NY, USA, 1999. ACM.
- [51] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.
- [52] Leo Sauermann. The gnowsis-using semantic web technologies to build a semantic desktop. Diploma thesis, Technical University of Vienna, 2003.
- [53] Leo Sauermann. The gnowsis — using semantic web technologies to build a semantic desktop. Master’s thesis, Vienna University of Technology, 2006.
- [54] Leo Sauermann, Andreas Dengel, Ludger van Elst, Andreas Lauer, Heiko Maus, and Sven Schwarz. Personalization in the EPOS Project. In *Proceedings of the Semantic Web Personalization Workshop at the ESWC 2006*, 2006.
- [55] Bernhard Schandl. SemDAV: A File Exchange Protocol for the Semantic Desktop. In *Proceedings of the Semantic Desktop and Social Semantic Collaboration Workshop*, volume 202, Athens, GA, USA, November 2006. CEUR Workshop Proceedings.
- [56] Bernhard Schandl and Ross King. The semdav project: metadata management for unstructured content. In *Proceedings of the 1st international*

- workshop on Contextualized attention metadata: collecting, managing and exploiting of rich usage information*. ACM Press New York, NY, USA, 2006.
- [57] Bernhard Schandl and Ross King. The SemDAV Project: Metadata Management for Unstructured Content. In *CAMA '06: Proceedings of the 1st International Workshop on Contextualized attention metadata: collecting, managing and exploiting of rich usage information*, pages 27–32, New York, NY, USA, 2006. ACM Press.
- [58] Ben Schneiderman. Dynamic queries for visual information seeking. *IEEE Softw.*, 11(6):70–77, 1994.
- [59] Schraefel, Daniel A. Smith, Alisdair Owens, Alistair Russell, Craig Harris, and Max Wilson. The evolving mspace platform: leveraging the semantic web on the trail of the memex. In *HYPertext '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 174–183, New York, NY, USA, 2005. ACM Press.
- [60] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface : Strategies for Effective Human-Computer Interaction (4th Edition)*. Addison Wesley, March 2004.
- [61] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.
- [62] Carolyn Snyder. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. Morgan Kaufmann, 2003.
- [63] Constantine Stephanidis, Demosthenes Akoumianakis, and Alex Paramythis. User Interaction in Digital Libraries: Coping with Diversity through Adaptation. *Lecture Notes In Computer Science*, 1513:717 – 735, 1998.
- [64] Diman Todorov and Bernhard Schandl. Small-scale evaluation of semantic web-based applications. Technical report, University of Vienna, 7 2008.
- [65] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.
- [66] Max L. Wilson and m.c. schraefel. mSpace: What do Numbers and Totals Mean in a Flexible Semantic Browser. In *Proceedings of the Semantic Web Personalization Workshop at the ESWC 2006*, 2006.

- [67] Ka P. Yee, Danyel Fisher, Rachna Dhamija, and Marti A. Hearst. Animated exploration of dynamic graphs with radial layout. In *INFOVIS*, pages 43–50, 2001.
- [68] Degi Young and Ben Shneiderman. A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation. *Journal of the American Society of Information Science*, 44(6):327–339, 1993.