

View-Based Search Interfaces for the Semantic Web

Eetu Mäkelä

Helsinki June 2, 2006

M. Sc. Thesis

UNIVERSITY OF HELSINKI

Department of Computer Science

Tiedekunta/Osasto — Fakultet/Sektion — Faculty		Laitos — Institution — Department	
Faculty of Science		Department of Computer Science	
Tekijä — Författare — Author			
Eetu Mäkelä			
Työn nimi — Arbetets titel — Title			
View-Based Search Interfaces for the Semantic Web			
Oppiaine — Läroämne — Subject			
Computer Science			
Työn laji — Arbetets art — Level		Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
M. Sc. Thesis		June 2, 2006	72 pages
Tiivistelmä — Referat — Abstract			
<p>This thesis explores the possibilities of using the view-based search paradigm to create intelligent search interfaces on the Semantic Web. After surveying several current semantic search techniques, the view-based search paradigm is explained, and argued to fit in a valuable niche in the field. To test the argument, OntoViews, a semantic view-based search portal creation tool was designed and implemented, and eight portals with five vastly different user interfaces were built using it. Based on the results of these experiments, this thesis argues that the paradigm, particularly as implemented in the OntoViews tool provides a strong, extensible and flexible base on which to built semantic search applications. The particular problems faced in applying view-based search for semantic interfaces are noted, along with explanations on how they were solved in the OntoViews architecture. Finally, directions and ideas for future research are presented for both the paradigm and the implementation architecture, respectively.</p> <p>ACM Computing Classification System (CCS):</p> <p>H.3.3 [Information Search and Retrieval]: Search process, Selection process, Query formulation</p> <p>H.5.2 [User Interfaces]: Theory and methods, Interaction styles</p> <p>I.2.4 [Knowledge Representation Formalisms and Methods]: Semantic networks</p>			
Avainsanat — Nyckelord — Keywords			
Semantic Web, Information Retrieval, View-Based Search, Multi-Facet Search			
Säilytyspaikka — Förvaringsställe — Where deposited			
Kumpula Science Library, serial number C-			
Muita tietoja — övriga uppgifter — Additional information			

Contents

1	Introduction	1
2	Semantic Web Technologies	1
3	Semantic Search Related Research	4
3.1	Research Directions in Semantic Search	4
3.1.1	Augmenting Traditional Keyword Search with Semantic Techniques	4
3.1.2	Basic Concept Location	6
3.1.3	Complex Constraint Queries	7
3.1.4	Problem Solving	9
3.1.5	Connecting Path Discovery	10
3.2	Common Methodology	10
3.2.1	RDF Path Traversal	10
3.2.2	Mapping Between Keywords and Concepts	11
3.2.3	Graph Patterns	11
3.2.4	Logics	11
3.2.5	Combining Uncertainty with Logics	12
3.3	Conclusions Drawn from the Survey	12
4	View-Based Search for the Semantic Web	13
4.1	The View-Based Search Paradigm	13
4.2	View Projection from Ontologies	15
4.3	View-Based Search as a General Base for Semantic Search Interfaces . .	17
5	Semantic View-Based Search Interfaces	18
5.1	A View-Based Search Interface for Browsing: MuseumFinland	19
5.2	View-Based Search in Your Pocket: MuseumFinland Mobile	27
5.3	A View-Based Search Interface for Efficient Search: Veturi	30
5.4	An Alternate Search/Browsing Interface: Orava	35

	iii
5.5 Applying View-Based Search for Ontology Browsing: ONKI	37
5.6 Adapting the Paradigm to Other Data	39
6 The Architecture of OntoViews	40
6.1 The Projection and Semantic Linking Engine Ontodella	42
6.1.1 View Projection Rules	42
6.1.2 Semantic Link Rules	44
6.2 The Semantic View-Based Search Engine Ontogator	45
6.2.1 Adaptability to variant domains	46
6.2.2 Interfacing with Other Semantic Components	48
6.2.3 Category Identification	48
6.2.4 Extensibility	49
6.2.5 Scalability	50
6.3 The Interaction and Control Component OntoViews-C	52
7 Discussion	55
8 Future Work	58
9 Conclusion	63
References	66

1 Introduction

The Semantic Web [BLHL01] is the vision of the W3C organization for the next generation of the World Wide Web, pushing for machine understandability of content. Currently, information is typically encoded in natural language in web pages. On the Semantic Web that information would be encoded in a format machines could “understand”, that is, process on the level of a formal semantic annotation of meaning.

While such a coding makes it possible for applications to intelligently process information, the formal semantics of the Semantic Web are not clear to an average human user. In user interface research, a core challenge here is in how to enable users to harness the power of the Semantic Web, while hiding the complexity.

This thesis explores the issue from the viewpoint of semantic search interfaces. First, a background primer on Semantic Web technologies is given in chapter 2. Then, in chapter 3, a survey of current semantic search related research is undertaken. Based on this, an argument is made in favor of the view-based search paradigm as a useful base for semantic search in chapter 4.

To test the value and extensibility of the paradigm, a portal creation tool called OntoViews was developed, and several view-based search interfaces built using it. Chapter 5 describes the results gained and the challenges faced while applying the tool to interface design, while chapter 6 deals with the implementation architecture itself. Chapter 7 contains discussion on the benefits and limits of the current approach, and chapter 8 lists possible directions for future work. The thesis ends by listing conclusions.

2 Semantic Web Technologies

The Semantic Web is based on encoding semantic-level information in a common formal way. To facilitate this, the Semantic Web relies on a common data model, and various semantics-specifying languages layered on top of it.

Underlying all else is the RDF data model, which specifies how information on the Semantic Web is to be represented. The model is based on a collection of simple triplets of the form (Subject, Predicate/Property/Relationship, Object), mimicking simple factual sentences such as “Finland/is a part of/Europe” or “Finland/is a/country”. In RDF, however, each subject and relationship used in a statement has a global and unique identifier, while the object can either be another such entity identifier, or a literal value. By using

the same identifiers in multiple triplets, a net of nodes and arcs is formed, linking the triplets together into graphs, and thus allowing for more complex forms of information to be modeled and stored.

An example of an RDF network is depicted in figure 1, describing some metadata about this thesis. In RDF, globally addressable entities are demarcated by URIs, in the figure shortened using the XML namespace notation [BHL99]. For resources that don't need a globally addressable identifier, anonymous blank nodes may be used. In this case, they are locatable only through their associations to other entities. In the example, the thesis itself is represented by such a blank node in the center of the graph. While this renders it impossible to directly address in an outside source, in self-contained applications this is still a common information encoding pattern, especially for nodes whose only role is to link together other information. In the example, the blank node is related by the property “e:author” to an individual, whose “e:name” is “Eetu Mäkelä”. The “e:title” of the object is “View-Based Search Interfaces for the Semantic Web”. It is “e:about” something referred to by the resource “e:semanticWeb”, as well as “e:about” something referred to by “e:search”.

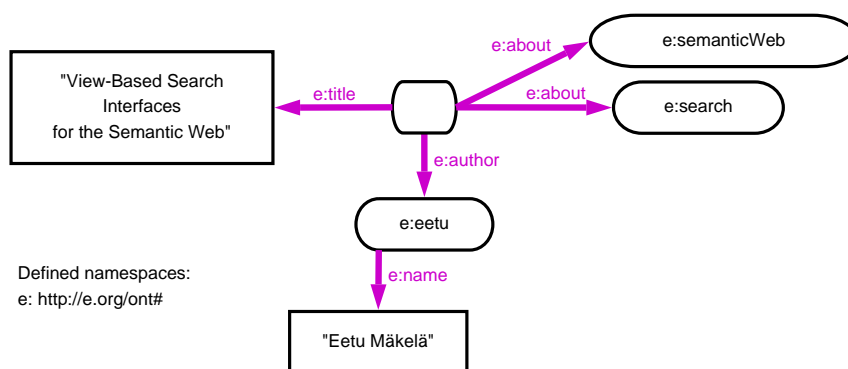


Figure 1: A visualization of an example RDF network

There are still more complexities in the RDF model, such as collections and containers that group resources together, as well as reification, where statements can refer to other statements. Also these constructs are represented using the triplet model. They are, however, not relevant to understanding this thesis, and thus will not be covered further here.

While the RDF data model provides a simple way to represent nearly any information, it does not generally specify what the concepts and relations used mean, what they entail. This is because RDF provides only a bare minimum of formal semantics [Hay04]. For example, in the graph of figure 1, there is still nothing telling the computer what the blank node actually is, or what “e:author”, “e:title”, or “e:semanticWeb” mean.

On the Semantic Web, the further formal semantics still needed are provided by ontologies, themselves defined using RDFS [BG04] and OWL [MvH04], the standard ontology languages of the Semantic Web. An ontology can be described as a formal system that describes some particular field of interest from the viewpoint of the ontology user. They are usually defined as a set of classes, concepts, properties, relationships, rules and restrictions.

A sample ontology continuing the previous example is depicted in figure 2. Here, it is learned that the resource “e:eetu” is a person, that the anonymous object is a thesis and the two other resources are topics. The relationships used are also present as instances of the class “owl:ObjectProperty”, and their possible domains and ranges defined.

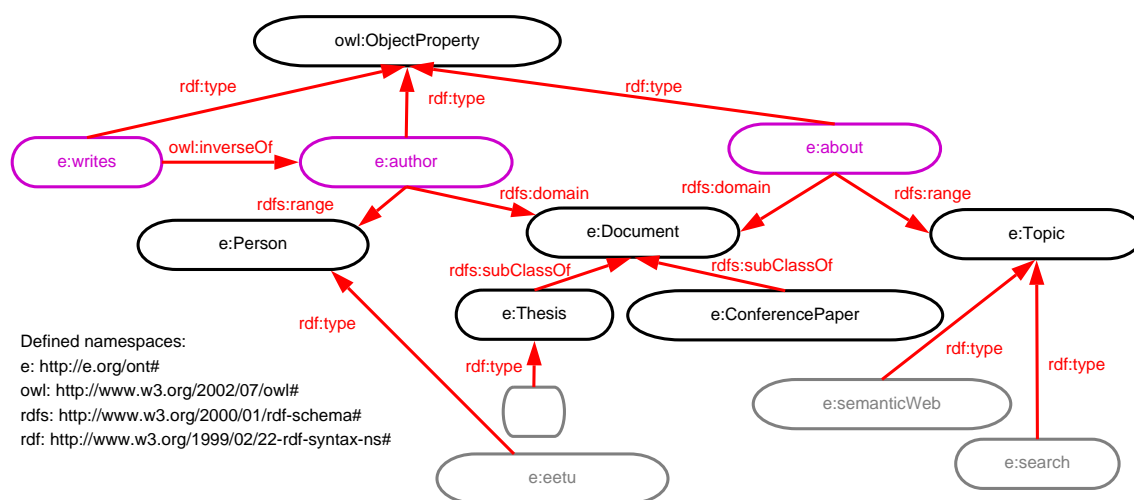


Figure 2: An example of an ontology

Also present, defined using the “rdfs:subClassOf” property, is a class subsumption hierarchy. Creating such a taxonomy is usually considered the first and most important step in ontology creation. This subsumption hierarchy also has semantic entailments defined in the underlying ontology language. For example, subclasses may inherit the various defined relationships of their superclasses. The “owl:inverseOf” property that has been defined between the properties “e:writes” and “e:author” can be used to do reasoning, too. Based on the existence of the property, the formal semantics of OWL define that for each triple of the form (X,”e:author”,Y), a triplet of the form (Y,”e:writes”,X) can be inferred. With data stored in the RDF data model, and with ontologies adding formal deduction capabilities to that data, a Semantic Web of smart data is formed. Having such smart data enables application designers to create intelligent applications more easily, as much of the intelligence needed is already encoded in the data.

3 Semantic Search Related Research

Because all the format, breadth and depth of the information on the Semantic Web differ from the current norm, the Semantic Web also poses new challenges in information retrieval. To understand what exactly these challenges are, a survey of current research related to the field was conducted. This section of the thesis presents the results of that survey, both to show the wider background context from which the developed approach emerges, as well as to provide a basis for comparison and evaluation.

For the survey, semantic search was defined as either search using semantic techniques, or search of formally annotated semantic content. The survey is based on reading and exploring some 25 different papers and approaches fitting that definition. The material was gathered based on a keyword search for “semantic AND search” in various publication databases, as well as by going through references in papers already found.

From the data gathered in the survey, some prevalent research directions in semantic search were identified, based on likeness of research goals. These, as well as the individual approaches that are part of them, are described in chapter 3.1. Besides research directions, the papers were also analyzed for common methodology. The methods used in a particular paper are noted when discussing it, but the descriptions of the common design patterns are presented in chapter 3.2.

3.1 Research Directions in Semantic Search

From the corpus of research used in the survey, five distinct research directions emerged. While the categories sometimes do not differ much in methodology, they seem separate and coherent enough on research goals to function as an informative clustering of the research space. The five directions are: augmenting traditional keyword search with semantic techniques, basic concept location, complex constraint queries, problem solving and connecting path discovery. All of these are described in detail in the following sub-chapters.

3.1.1 Augmenting Traditional Keyword Search with Semantic Techniques

Much, particularly early research on Semantic Web enabled search deals with augmenting traditional text search with semantic techniques. This research direction differs significantly from the others presented later in the sense that it does not usually presume most of the knowledge being sought to be formally annotated. Instead, ontological techniques

are used in a multitude of ways to augment keyword search, whether to increase recall or precision.

Many query expansion implementations used in keyword search make use of thesaurus ontology navigation as a step in query expansion. Particularly used is the large WordNet [Fel98] ontology, defining synonym sets for words. The systems work as follows. First, the keywords are located in an ontology. Then, various other concepts are located through graph traversal. Finally, the terms related to those concepts are used to either broaden or constrain the search. In [MM00] and [BRA05], terms are expanded to their synonym and meronym sets using the Boolean OR operations available in most search engines. In Clever Search [KNRK05], a particular meaning of a word in the WordNet ontology can be selected, resulting in the clarification text of that meaning being added to the search keywords with the Boolean AND operator. In the ontology navigation phase, the implementations differ mostly in which properties of the ontology are navigated and which terms are picked.

A simple manner of augmenting keyword search results is taken in the “Semantic Search” interface [GMM03] of the TAP infrastructure. Here, besides a traditional keyword search targeted at a document database, the keywords are matched against concept labels in an RDF repository. Matching concepts are then returned alongside the found documents. The paper also proposes a continuation of the search similar to Clever Search [KNRK05], where, if multiple concepts match the keyword, the user can select his intended meaning to constrain the search. Here, however, the idea is not to expand search terms, but to constrain results based on existing semantic annotations concerning them.

[RSdA04] describes an algorithm for locating extra information relevant to a query given a starting set. First, traditional text search is applied into a document collection. Then, a process of RDF graph traversal is begun from the annotations of those documents. The intent is to find concepts related to the result, such as the writer of the document or the project the document refers to in a general manner. The traversal is done by a spread activation algorithm, for the use of which the arcs in the ontology are weighed according to general interestingness. This interestingness measure is calculated by combining a specificity measure favoring unique connections in the knowledge base, and a cluster measure, which favors links between similar concepts.

The CIRI [AJS⁺04] search system provides an ontological front-end to text search. The search is done through an ontology browser that visualizes the ontologies created for search as subsumption trees, from which concepts can be selected to constrain the search. The actual search is done through keywords annotated to these concepts as well as any

subconcepts, using a traditional text search engine and Boolean logic. The search algorithm is in many ways similar to the query expansion algorithms discussed before. The main difference is in the user interface being based on direct ontological browsing, leaving out the first step of mapping a search keyword to the ontology.

3.1.2 Basic Concept Location

While much of semantic search research is directed at adding semantic annotations to data to improve search precision and recall on that data, there are other reasons for writing down information with formal semantics. Therefore, some research begins with assuming concepts, individuals and relationships, and deals with the task of efficiently finding instances of these core Semantic Web datatypes.

Usually, the data the user is interested in are individuals belonging to a class, but the domain knowledge and relationships are described mainly as class relationships in the ontology. This organization of data points to a natural way of locating information, represented for example in the SHOE [HH00] search system. In SHOE, the user is first given a visualization of the subsumption tree of classes in the ontology, from which he can choose the class of instances he is looking for. Then, the possible relationships or properties associated with the class are sought, and a form is presented that allows the user to constrain the set of instances by applying keyword filters to the various instance properties. When the properties point to objects, the target of the filtering will be the label of the referenced resource. Queries that can be expressed using this paradigm are for example “find all publications with a particular author name, from a particular project”. A similar approach is also taken in the ODESeW [CGPLC⁺03] portal tool.

A major drawback of the approach is that ontological knowledge is only used to produce a keyword form, and the user is still left to guess what keywords will result in the instances sought. This can be averted if the database is built in such a way that there are not too many items in a category, so they can be all shown for visual inspection. This approach is taken in the many Internet directories such as the Open Directory Project directory¹ and the Yahoo! directory², where the editors are tasked with pruning the items and creating a branching category tree to hold them.

Once the search has advanced to the point where at least a single interesting instance is found, more information can be retrieved by browsing. The process is analogous to

¹<http://www.dmoz.org/>

²<http://dir.yahoo.com/>

browsing web page hyperlinks. However, here the items shown are resources and the links between them are defined by their relations. In the simplest case, one concept is shown at a time, with its properties taken straight from the RDF triples. If a property points to another resource and not a literal, then clicking on that property will browse to the referenced concept. This is the approach taken for example in the SEAL portal tool [MSS⁺01].

The authors of the Haystack information management tool [KBH⁺05, QHK03] base their user interface paradigm almost completely on browsing from resource to resource. They argue this by search behavior research [TAAK04], concluding that most searching is done by means of a process called orienteering. The premise is that searchers usually don't actually themselves know or remember the specific qualities of what they are looking for, but have some idea of other things related to the sought item. The process of search is then a browsing experience in which the searcher looks for information resources that he knows are somehow related to the target. This continues iteratively, until enough additional information on the target resource has been found, and it can be located.

An example in [TAAK04] is of a person searching for a particular piece of documentation. Not remembering where it is stored, she only remembers that it was referenced to in some e-mail message from a co-worker. She then scans through her mails in her inbox and, remembering the co-worker who the mail was from, finds the correct message and from there extracts the location of the relevant document. To ease finding points of entry for orienteering, Haystack provides a simple text search interface, based on the rationale that the things people remember about resources are probably their labels or phrases contained in them.

3.1.3 Complex Constraint Queries

Many kinds of complex queries can be formulated as finding a group of objects of certain types connected by certain relationships. On the Semantic Web, this translates to graph patterns with constrained object node and property arc types. An example would be "Find all toys manufactured in Europe in the 19th century, used by someone born in the 20th century". Here "toys", "Europe", "the 18th century", "someone" and "the 19th century" are ontological class restrictions on nodes and "manufactured in", "used by" and "time of birth" are the required connecting arcs in the pattern.

While such patterns are easy to formalize and query on the Semantic Web, they remain problematic because they are not easy for users to formulate. Therefore, much of the re-

search in complex queries has been on user interfaces for creating complex query patterns as intuitively as possible.

[ACK04] presents GRQL, a graphical user interface for building graph pattern queries based on navigating the ontology. First, a class in the ontology is selected as a starting point. All properties defined as applicable to the class in the ontology are then given for expansion. Clicking on a property expands the graph pattern to contain that property, and moves selection to the range class defined for that property. For example clicking the “creates” property in an “Artist” class creates the pattern “Artist → creates → Artifact”, and moves focus to the Artifact class, showing the properties for that class for further path expansion. The pattern can also be tightened to concern only some subclasses of a class, as in tightening the previous example to “Artist → creates → Painting or Sculpture”. In a similar way, property restriction definitions can be tightened into subproperties. More complex queries can be created by visiting a node created earlier and branching the expression there, creating patterns such as the one visually depicted in figure 3. This pattern could be used to find all artists that have either created any sculptures, or paintings good enough to be exhibited at a museum, as well as those sculptures, paintings and museums.

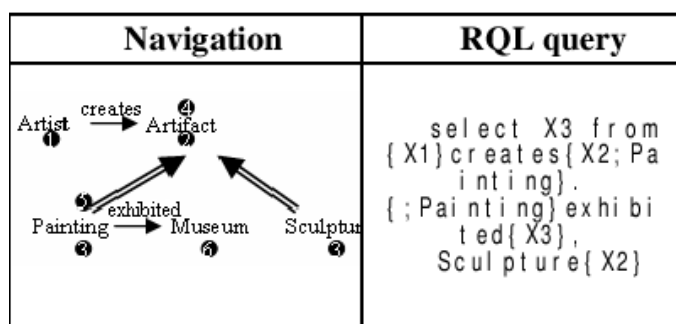


Figure 3: A visual formulation of a query in the GRQL interface, along with the generated query language expression

Another graphical query generation interface is the SEWASIE visual tool for query formulation support [CDM⁺04]. Here, the user is given some prepared domain-specific patterns to choose from as a starting point, which they can then extend and customize. This is done through a clickable graphic visualization of the ontology neighborhood of the currently selected class, as shown in figure 4. The refinements to the query can be either additional property constraints to the classes, for example “Industry with sector Agriculture” or a replacement of a class in the pattern with another compatible one, such as a sub- or superclass.

All of the individual constraints in a complex semantic query need not be ontological.

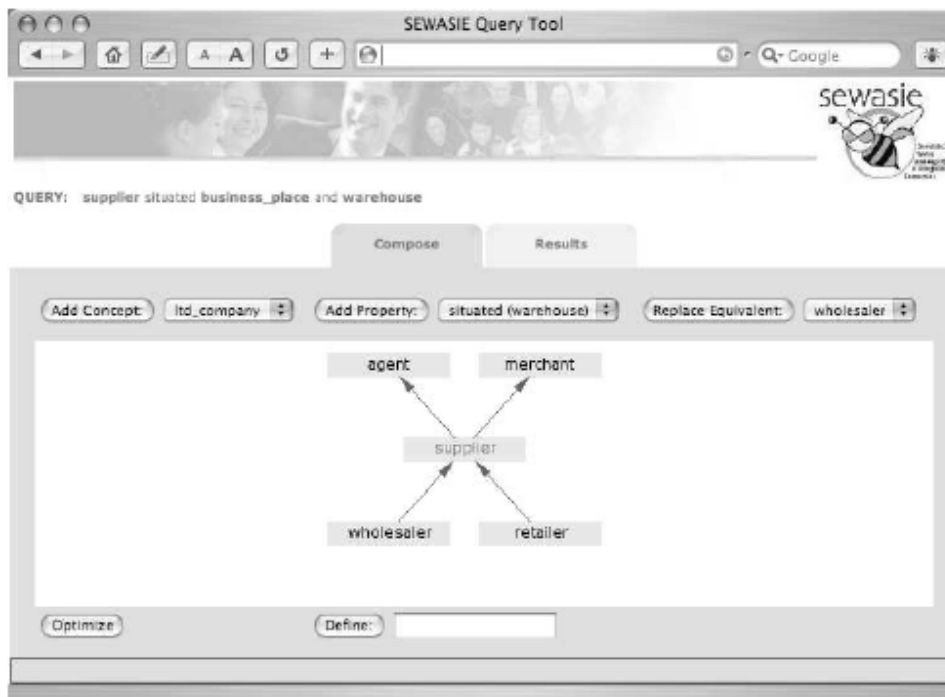


Figure 4: The SEWASIE visual tool for query formulation support

[ZYZ⁺05] contains a method that allows one to treat keyword search terms as ontological classes whose instances have fuzzy membership values. A fuzzy logic formalism is then used to calculate relevance with respect to the entire query pattern formalized as a fuzzy logic statement.

3.1.4 Problem Solving

Describing a problem and searching for a solution by inferring one based on ontological knowledge is one of the core use cases often associated with the vision of the Semantic Web. However, current implementations are rare.

An example is the Wine Agent demonstration portal [HM03]. Here, the user enters information on the flavors in a dish, and the system infers a recommendation for a wine suitable to complement those flavors. The service is primarily based on restrictions and knowledge directly encoded in the OWL ontology of the portal. When a query comes in, a general purpose Description Logic reasoner is employed to perform constraint satisfaction on a combination of knowledge in the query and knowledge in the ontology. To encode the prerequisite knowledge in the query, the SQL-like query language OWL-QL [FHH03] was developed.

3.1.5 Connecting Path Discovery

While usually property relations are used to traverse from an interesting resource to another, sometimes what is interesting are the connecting paths themselves. In the realized vision for the Semantic Web, a huge amount of varied semantic data will be available to be mined for semantic connections. An example of a domain where this could prove useful is the national security domain, where there is a need for finding, for example, emerging links between known terrorists and potential recruits [AS03].

A major problem here is how to define a measure of link interestingness in a way which cuts out uninteresting relations but is still general enough to be of use in finding complex, hidden relationships in the data. For example, “Company A and terrorist organization B are related because they both operate in the same country” is a conclusion, but not an interesting one. [AS03] presents one take on the problem, attempting to draft an easily calculable general purpose requirement for interesting associations.

3.2 Common Methodology

In surveying semantic search related research some common methodologies appeared. Some are inherent to the RDF formalism and will probably be present in all Semantic Web -based applications, while others are more tied to the search domain. Identifying and understanding these common methods and how they are used in the various actual approaches provides valuable background for devising and evaluating new approaches, such as the view-based approach presented in this thesis.

3.2.1 RDF Path Traversal

Because the data model of RDF is a graph, where arcs and multiple arc paths encode information, it is natural to apply graph traversal in semantic search.

There are a few primary uses of network traversal found in this survey. One is finding more relevant information instances given a starting instance in the net, as in [RSdA04]. Another use is in query formulation, such as in the GRQL [ACK04] and SEWASIE [CDM⁺04] interfaces, where a query is constrained by navigating the classes and relationships.

Simple path traversal is also usually used when gathering all the information about an item for visualization. This is again because of the way the RDF data model works: in-

formation important to the user is also found in other resources linked to an information item, and not just the direct properties of that item. At least SEAL [MSS⁺01] and Semantic Search [GMM03] both make use of graph patterns for gathering the information to be shown for an item.

3.2.2 Mapping Between Keywords and Concepts

Mapping between keywords and formal concepts is a common pattern appearing in semantic search. There are several reasons for its prevalence. The first is that a presupposition of all knowledge sought being formally encoded is blindly optimistic. Much research, such as the fuzzy keyword to concept mapping of [ZYZ⁺05], is specifically about how to combine searching through textual material with search through formally defined information.

A second reason is that natural language is the form of expression that comes most naturally to humans. Mapping patterns in the graph to sentences, such as in the SE-WASIE visual query tool [CDM⁺04] can give the user a clearer picture of what the relationships represent. On the other hand, the user may be more comfortable in expressing their queries as natural language sentences, as in the WordNet-based systems [MM00, BRA05, KNRK05].

3.2.3 Graph Patterns

Whether described in RDF path or logical languages, graph patterns are an important concept in semantic search, used in multiple different roles. First, graph patterns are often used to formulate and encode complex constraint queries as discussed in chapter 3.1.3, specifying and locating interesting subgraphs in the RDF network. In [AS03], general RDF patterns were also used to find interesting connecting paths between named resources. In result visualization, the specifications on where to fetch information relevant to the item are also usually given as graph patterns.

3.2.4 Logics

Logics and inference are integrally tied to the larger vision of the Semantic Web. For example, the web ontology language standard OWL [MvH04] is based on Description Logics. However, only few applications are currently built solely on top of these capabilities, with the Wine Agent [HM03] being an exception rather than a common example.

Much more commonly, applications make use of a few particular entailments as a base, and build their own functionality on top of that. For example SHOE [HH00], ODESeW [CGPLC⁺03], GRQL [ACK04] and SEWASIE [CDM⁺04] all make use of the transitive subClassOf hierarchy, and some also the properties conferred to a class by that hierarchy.

3.2.5 Combining Uncertainty with Logics

In the research direction of augmenting text search with ontology techniques, there is a need for formalisms which allow for combining uncertain annotations based on text search with the firmness of semantic annotations. As a result, several formalizations for, and experimentations with fuzzy or probabilistic logics, relations and fuzzy concepts have been undertaken in that field. The method described in [ZYZ⁺05] is an example.

Fuzzy logics are, however, not only useful in combining text search with ontologies. On the search method research side not directly tied to actual applications, [SDA04] applies fuzzy qualifiers to complex constraint queries. In [Par04], the idea is presented that user profiling could be used as a basis for weighting the interestingness of an ontological relation to be used in the search. In [KH05], a basis is depicted for calculating overlap values for historical and current geographic places, for use in a probabilistic mapping of the concepts to one another in any ontological search.

3.3 Conclusions Drawn from the Survey

There are many common patterns found in the approaches described in this survey. On the technique level, it seems that many of the methods used are general, separable modular steps. They could probably be used in most of the systems, regardless of research direction or application domain.

It also seems that some of the research directions can be combined. First, simple concept location can be seen as a forerunner and subset of the interfaces allowing selection by more complex graph patterns. Second, while the current interfaces for creating graph query patterns concern fairly simple patterns where the individuals and classes are the interesting information items, there is no theoretical reason for such a limitation. Because relations appear as equal partners in the underlying data model, querying for them would only need a shift in focus on the query formulation user interface level. Fuzzy logic formalisms and fuzzy concepts would allow for the inclusion of keyword search results in the queries. After finding a result set using complex constraints, graph traversal algorithms could be applied to find additional result items.

The only direction that does not neatly wrap into the others is pure inference-based problem solving. However, as already stated, many of the applications do make use of the logical entailments in one form or another, they only do not rely on them completely.

4 View-Based Search for the Semantic Web

Based on the conclusions drawn above, it seems that complex graph matching patterns form a useful, extensible technology core for semantic search. However, as stated, a major challenge in using it is in how to provide the end user with an intuitive interface for creating graph-based queries. The rest of this thesis is based on the argument that the so-called view-based search paradigm [Pol98] provides a suitable basis for creating such interfaces, after first explaining the paradigm, below.

4.1 The View-Based Search Paradigm

The core idea of view-based search, also called multi-facet search, is to provide multiple, simultaneous views to an information collection, each showing the collection categorized according to some distinct aspect. This is based upon a long-running library tradition of faceted classification [Map95]. A search in the system then proceeds by selecting subsets of values from the views, constraining the search based on the aspects selected.

The paradigm was first developed into a computer application in the HiBrowse [Pol98] system for searching through large collections of medical texts. Figure 5 depicts the interface of HiBrowse as an example of what view-based search can look like for an end user. After HiBrowse, the idea of view-based search has been implemented in a number of systems. Usability studies done on these systems, such as Flamenco [LSLH03, EHS⁺03, HEE⁺02] and Relation Browser++ [ZM05] have proved the paradigm both powerful and intuitive for end-users, particularly in drafting more complex queries. More evidence suggesting the power of the paradigm comes from more general results on the benefit of using multiple categorizations in search [Hea00, QBHK03].

In all current view-based search systems, the views used are either flat or hierarchical tree categorizations of the search items. There are several good reasons for using such views. First, such categorizations are familiar to users, from for example library classification systems. Second, they can often be drawn up from any aspect of a collection, which allows for a uniform look and feel for the views. In this thesis, one reason for favoring tree categorizations also relates to how the paradigm is combined with the Semantic Web

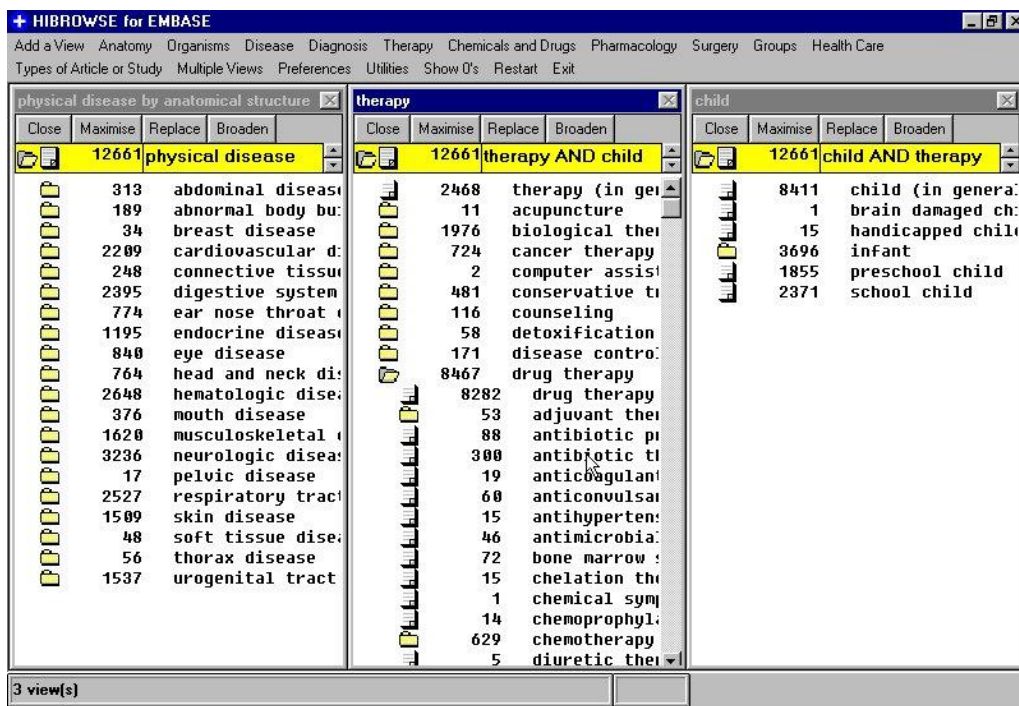


Figure 5: The HiBrowse interface, with three hierarchical views

ontological hierarchies, described later.

Figure 6 shows a conceptual overview and an example of view-based querying using hierarchical categorizations. Here, on the left, the data representing a museum collection of items has been categorized according to three hierarchical views: “Location of Manufacture”, “Item Type” and “Location of Use”. The idea of view-based search, then, is that given these views, the user can apply multiple constraints on any of the views, with the effects of filtering immediately shown in all the views. Simultaneous constraints in different views are applied by simply performing an intersection operation on the results of the constraints in each view. In the example of figure 6, the user has selected as query constraints the category “Office Equipment” from the “Item Type” facet, and the category “Finland” from the “Location of Use” facet. Intuitively, the user is searching for any office equipment in the collection that happens to have been used anywhere in Finland.

Inside a hierarchical view, the constraint is calculated as follows. When the user selects a category c in a view v , the system constrains the search by leaving in the result set only such objects that are annotated in view v with some subcategory of c or c itself. In the figure, this is typified in the “Location of Use” view. Here, none of the objects are directly annotated as belonging to the category Finland, but some are nonetheless taken as matching, based on the implicit knowledge in the category hierarchy that Lahti and

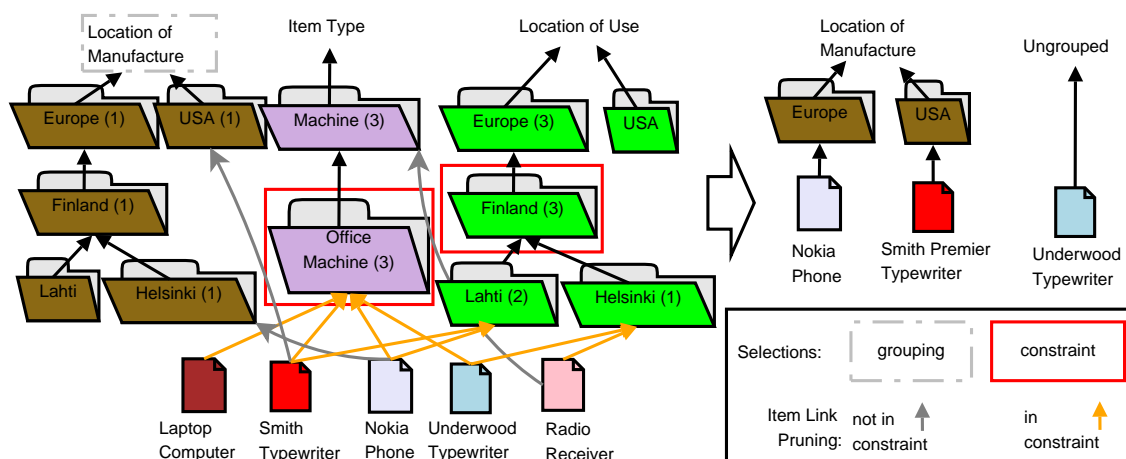


Figure 6: A conceptual overview of view-based querying

Helsinki are located in Finland.

A core idea of view-based search is that once the result set is calculated, it is categorized according to the views and visualized in place. This can be done for example by showing the number of results in each view category beside them, as in figure 6 and the HiBrowse interface in figure 5. The result of applying this idea is a tight, beneficial relationship between query constraining and result browsing. First, the user is immediately able to gauge the result set from multiple different aspects. Second, the user is given direct, accurate information on how any further selections will limit the result set. The system can also directly cut out category choices with no associated results as further selections, because selecting them would lead to an empty result set.

In addition to in place visualization, separate views suited particularly to organizing the results of a search can be used. For example, on the right in figure 6, a flat column result grouping has been formed using the “Location of Manufacture” category tree. This has been accomplished by cutting the hierarchy on the first sublevel, and sorting the result items into these categories. Item “Nokia Phone” is bumped two levels up to its ancestor category of “Europe”, and item “Underwood Typewriter”, which was not annotated anywhere within the grouping hierarchy, is shown within the dynamically created “un-grouped” category.

4.2 View Projection from Ontologies

In non-semantic view-based search systems, the focus on hierarchical views was brought by the prevalence of taxonomical classification systems in the collections the systems

were built for. On the Semantic Web, domains are described more richly using ontologies. However, hierarchical hyponymy and meronymy relationships are still important for structuring a domain. Therefore, these ontologies typically contain a rich variety of such elements, most often defined with explicit relations, such as “part-of” and “subclass-of”. This naturally leads to the idea of using these hierarchical structures as bases for views in view-based searching. To carry this out, this chapter introduces a process termed view projection. Here the process is explained in abstract terms. Details of the actual systems produced are found later, in the implementation part of this thesis.

An example of view projection using the process is given in figure 7. The transformation described consists of two important parts: projecting a view tree from the graph, and linking items to the categories projected. The projection of a hierarchical category tree can be done through traversing the graph by some rule, picking up relevant concepts and linking them into a tree based on the relations they have in the underlying knowledge base. Most commonly, the relations used are hyponymies and different kinds of meronymies.

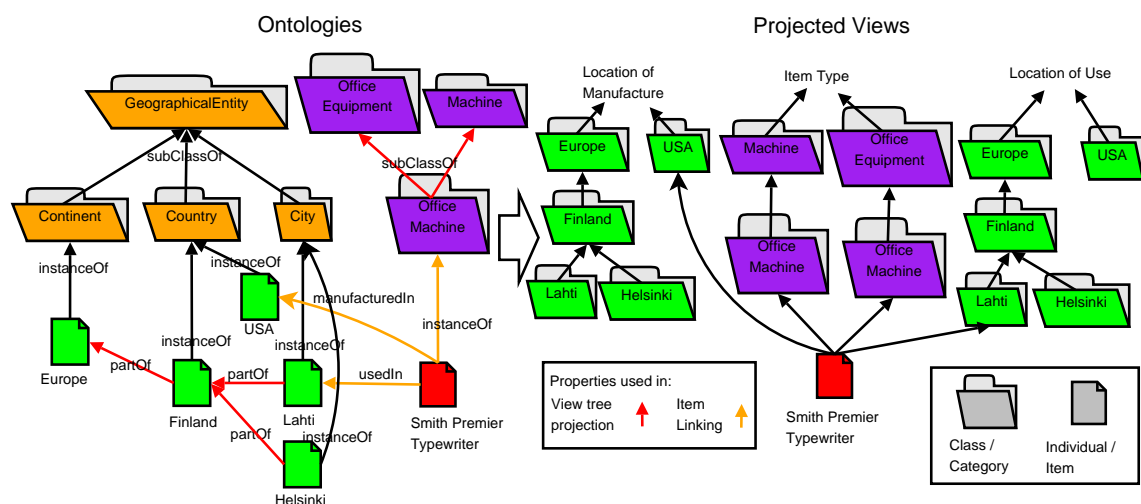


Figure 7: An example of view projection

In the example, the “Item Type” view is projected using a simple rule following the “sub-ClassOf” hyponymy relationship, starting from a pair of selected roots. The rules governing projecting the “Location” meronymy tree are a little more complex. It is created by taking all instances of the class “GeographicalEntity” and its subclasses, but then creating a category tree from these instances by traversing their “partOf” relationships.

In projecting a tree from a directed graph, there are always two things that must be considered. First, possible loops in the source data must be dealt with to produce a Directed Acyclic Graph (DAG). This usually means just dismissing arcs that would form cycles in

the projection process. Second, classes with multiple superclasses must be dealt with to project the DAG into a tree. Usually such classes are either assigned to a single superclass or cloned, which results in cloning also the whole subtree below. In the example, the class “Office Machine”, in the “Item Type” view is cloned based on this rule.

The second phase of view projection is associating the actual information items with the categories. Most often, this is just a simple case of selecting a property that links the items to the categories, but it can get more complex than that here, too. As can be seen from the example in figure 7, the same hierarchy can also form the basis of several views, based on how linked items are selected. The geographical part-of hierarchy is projected into two views, based on whether the “usedIn” or the “manufacturedIn” relationship between the items and places is used. As an example where the item linking would be more complex, consider a view categorizing items based on the type of geographical entity they were manufactured in. Here, creating the view hierarchy would be a simple case of transitively following the “subClassOf” property of the class “GeographicalEntity”. However, both a “manufacturedIn” and an “instanceOf” property would have to be traversed to link the items to the categories.

4.3 View-Based Search as a General Base for Semantic Search Interfaces

The previous subsections showed a way of combining view-based search with the Semantic Web. However, there are still other requirements to be met before the paradigm can be considered useful as a general base for semantic search interfaces.

First, and most importantly, the interfaces created using the paradigm should be usable by an end-user for the tasks they need to perform on the Semantic Web. Usability studies [LSLH03, EHS⁺03, HEE⁺02] suggested the paradigm particularly useful for intuitively formulating complex queries. This, combined with the conclusions about complex queries forming a good technology core for semantic search intimate good results, provided the expressive power of the paradigm is sufficient.

View-based constraints can be seen as a limited form of complex graph constraints. At first sight, the formalism may seem restrictive compared with the more complex graph patterns formed by the interfaces presented in section 3.1.3 of the survey section. Widening the expression power of the approach, however, is the fact that the views can be complex projections of rich ontologies. It seems that most combinatorial constraints needed can be covered by choosing the views intelligently. The difference becomes that in view-based

search, much work must be done in figuring out the useful views and projecting them from the underlying ontology. However, a similar operation will probably prove necessary for the other formalisms as well, as already apparent for example in the preselected starting point queries of the SEWASIE [CDM⁺04] system.

Concerning projection, the formalism should be tested on adaptability to a wide range of different ontological data. It should also be easy to extend the paradigm itself to make powerful use of the rich semantics of that data. There are few inherent restrictions here. The only real requirement of a view is that it organize the information items of the application in some visualizable and constrainable way. Therefore, it should be quite possible to extend the paradigm to make use of other supporting semantic search methods.

While the above considerations point to a good potential for view-based search on the Semantic Web, the hypotheses still need real world verification. Combining all the requirements, the paradigm should make it possible to create powerful, efficient interfaces for varying search tasks aimed at real world ontological data. To test this, the view-based semantic portal creation tool OntoViews was created, and several portals built using it. The rest of this thesis concerns itself with the results of these experiments. First discussed are the interfaces created. Then, the architectural decisions made while designing the system are described.

5 Semantic View-Based Search Interfaces

To test the ability of the view-based paradigm to create powerful, intelligent search applications for the Semantic Web, five different semantic search -based web portal interfaces were created.

The applications designed were intended to span the range of information retrieval tasks. To aid in this, the various strategies identified in recent search behavior research [SMS02, TAAK04] were partitioned into two groups, designed to demarcate two different polar ends of search behavior. These groups are respectively termed “browsing” and “spot search”.

The browsing search strategy is characterized by the absence of a particular clear information need. Instead, the user is either looking to get an overview of some topic, or just looking for something interesting to explore. This agglomeration mostly contains the information gathering and browsing strategies identified in [SMS02].

Spot searching, the second strategy defined, relates closely to the finding behavior of

[SMS02], as well as the teleporting strategy defined in [TAAK04]. It is characterized by the need for a particular, singular piece of information without much regard to its context, and by the need to get it quickly.

It is argued here that the view-based search paradigm can adequately respond to both of these, in many cases opposite needs of searching and browsing. Additionally, there is value in being able to support them both at the same time. This is proved by the results of research into the prevalence of the orienteering search behavior, where these two modes are used intermittently [TAAK04]. Here the tight relationship between result browsing and query constraining in view-based search is an asset.

In the following, interfaces aimed mostly for browsing are first presented, followed by an interface developed mostly for spot search. After that, two other interfaces are presented. The first of these deals with a simplification of the browsing interface for a particular task, while the second presents the simple special case of using the paradigm for ontology browsing itself.

5.1 A View-Based Search Interface for Browsing: MuseumFinland

When a user's information need is not articulated, either because expressing that need is difficult or because the exact search goal is not well-defined even in the user's mind, collection browsing is a useful way of getting to know the available data sources. Using view-based search in such a situation was studied in the MuseumFinland³ portal [HMS⁺05], created in the "Semantic Web: Intelligent Directories"⁴ project. The portal demonstrates how the Semantic Web can be used to combine material from heterogeneous sources. It presents a collection of museum items from three museums, each originally having differing legacy database systems. To create the portal, metadata about collection items was extracted from these databases into RDF/XML, after which the annotations were semantically enriched [HSJ04].

As a result of this semantic annotation process, the items were linked to a set of seven RDFS ontologies. From these, nine different view hierarchies were projected in the user interface. Table 1 shows these views, the ontologies they are based on, as well as a grouping that relates them to four more general aspects of the items. Of particular interest is the "Situation of use" facet. Such a view seems natural and interesting when looked at from the viewpoint of a user browsing an exhibition, but has not traditionally been used

³The portal can be found at <http://www.museosuomi.fi/>, and includes an English online tutorial.

⁴<http://www.seco.tkk.fi/projects/semweb/>

as a classification scheme during indexing. As a result, it has seldom been seen in search engines so far. Another reason for this may be that the categorization does not seem so useful alone. However, here, with the power of view-based search it is easy to include the categories into useful combinations, such as “Asian instruments of war”, or “agricultural items from the 19th century”.

Item aspect	Facet view	Underlying ontology
Artifact	Artifact type (Esinetyyppi)	Artifacts
	Material (Materiaali)	Materials
Manufacture	Manufacturer (Valmistaja)	Actors
	Location of manufacture (Valmistuspaikka)	Locations
	Time of manufacture (Valmistusaika)	Times
Use	User (Käyttäjä)	Actors
	Location of use (Käyttöpaikka)	Locations
	Situation of use (Käyttötilanne)	Situations
Museum	Collection (Kokoelma)	Collections

Table 1: The nine view facets in the MuseumFinland portal with Finnish translations, and the seven domain ontologies they are based on.

Tree hierarchy	Sample categories
Artifacts	Transportation devices and their parts, Guns and shooting equipment
Materials	Building materials, Foodstuffs
Actors	Individuals, Institutions
Locations	Asia, Unspecified foreign
Times	Centuries, Ages
Situations	Celebrations and ceremonies, Food preparation
Collections	Espoo city museum collections, National museum collections

Table 2: Sample categories in the seven types of view hierarchies

The user interface of MuseumFinland owes much to the user interface studies conducted on the Flamenco system for locating fine arts images [LSLH03, EHS⁺03, HEE⁺02]. Figure 8 shows the opening view of MuseumFinland. The nine views (in Finnish, refer to table 1 for view name translations and table 2 for examples of categories) are shown spanning the whole screen. They are grouped into the four item aspects of table 1 visually, using color and positioning as hints. For each hierarchical view, the next level of subcategories is shown as a set of links under its heading. A category is added to the constraints

by clicking on its name. Only categories whose selection will not lead to an empty set are presented for selection. Beside each category is the number of items that would result if the user was to select it as a query constraint. The intent of this layout is to allow for the user an early overview of the collections being virtually exhibited, as well as the means they can be found with. In the primary usage scenario here the user does not know what she is searching for. Therefore, it is important to give her as many choices as possible for exploration. The intent is that at least one link in the many views will be found interesting and selected.

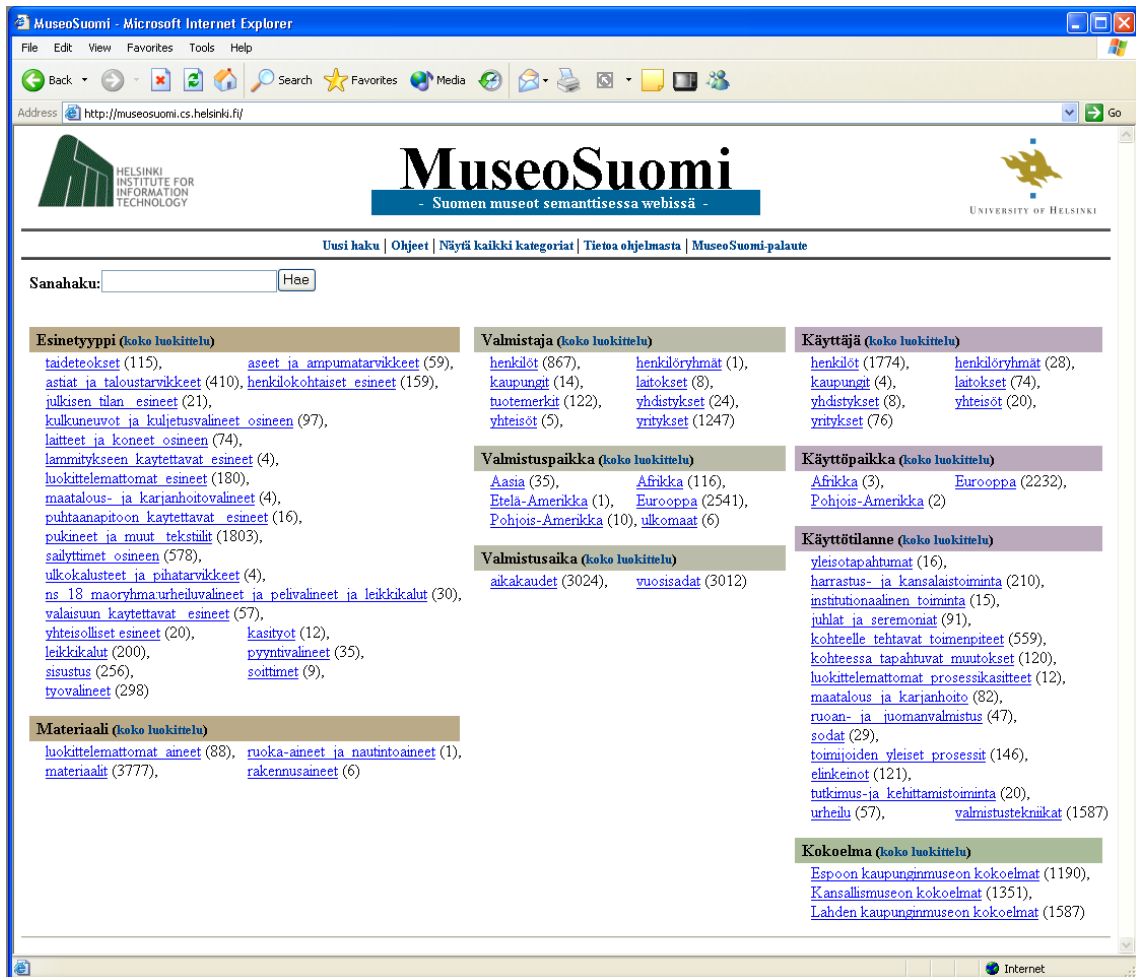


Figure 8: The initial search view of MuseumFinland

On selecting a category, the view changes. Figure 9 depicts the user interface after selecting the category “Tools” (“työvälineet”) from the “Artifact type” view. The views move to the left of the screen to make room for a visual rendering of the results, shown on the right. The view where the selection was made changes to show possible choices on the next sublevel. By default, the results are shown grouped by the direct subcategories of

the last selection, here for example “Textile tools” (“tekstiilityövälineet”) and “Tools of folk medicine” (“kansanlääkinnän työvälineet”). Hits in different categories are separated by horizontal bars and can be viewed page by page independently in each category. The number of hits shown in each subcategory is determined from the number of subcategories in the result set, in order to fit a maximum of useful information into the limited screen space. In the case of the example, there are so many groups that only a single line of hits is shown. Currently effective constraints are shown alongside the views, as well as on the top right of the page (’Hakuehdot’). The search can be relaxed from either location, if necessary.

Figure 9: The main search view of MuseumFinland

Further constraints can be selected either from the view facets, or also by clicking the header of any group of objects that looks interesting. At all times, the user can firmly

gauge the effects of possible choices by looking at the number of hits associated with the categories, and the user interface eliminates selections leading to empty result sets completely. On the whole, this interaction strategy where only one further level in each view is shown at a time produces an iterative approach to query constraining. While it does require many clicks to locate a particular item this is intentional. This is because the strategy is chiefly aimed for cases in which the user does not exactly know what they are searching for. The interaction behavior quickly gives the user an impression for what is contained in the portal collection, and provides to the user in each step a manageable set of choices to choose from. For example, looking at the main page of MuseumFinland, the user, not really looking for anything particular, may decide that she will start with looking at items used in Europe. In the results, she then sees several chairs she likes, and decides to constrain her search to furnishing items used in Europe, and so on.

Besides the default grouping based on last selection, the system also supports grouping along arbitrary views by clicking on the “Group targets” (“ryhmittele kohteet”) link next to each facet. Items in the result set that do not belong in any of the groups are gathered in an “Other hits” group. This can be useful in getting different overviews of the collections. For example, in the situation of figure 9, grouping by Museum Collection would provide the user a quick and intuitive view on what tools there are in each collection of the participating museums.

Showing only one flat level of each hierarchical grouping supports the interaction pattern wanted. However, sometimes this limits the overview gained in a harmful way for answering questions about the result set as a whole. The user interface of MuseumFinland therefore also provides an alternate view to the material and the facets of the application. Clicking the link “Whole facet” (“koko luokittelu”) on any facet brings up a tree view of the whole facet with the number of items in each category calculated according to current constraints. The tree view gives the user an overview of the distribution of items in the result over a wished dimension. By judicious use of this view, complex questions about the result set can be answered. For example, if a collection manager wants to know how well their collection covers tools manufactured at different ages, they can select the “Time of manufacture” whole facet view after constraining the query as described before. The resulting display is shown in figure 10. From the result and the visual cues, such as gray-ing out categories with no hits, it is easy to see several things. For example, while there is a balance in items relating to the two world wars (“I maailmasota” with 11 items and “II maailmansota” with 9 items), there are no items from 1700-1749 and only one from 1750-1799. Also, there are two items that could only be reliably dated as being manufactured at some point in the 18th century, explaining the total of 3 items for the category

“1700-luku”.

Hakuehdot

Kategoria: Esinetyyppi > *työvälineet* (ryhmittele kohteet) (poista)

Kategoria: Valmistusaika

- [aikakaudet](#) (90)
 - [esihistoriallinen aika](#) (1)
 - [kivikausi](#) (1)
 - [historiallinen aika](#) (89)
 - keskiaika
 - [uusi aika](#) (89)
 - [sodat](#) (19)
 - [I maailmasota](#) (11)
 - [II maailmansota](#) (9)
- [vuosisadat](#) (89)
 - 1600-luku
 - [1700-luku](#) (3)
 - 1700-1749
 - [1750-1799](#) (1)
 - [1800-luku](#) (20)
 - [1800-1809](#) (3)
 - [1810-1819](#) (2)
 - [1820-1829](#) (3)
 - [1830-1839](#) (4)
 - [1840-1849](#) (2)
 - [1850-1859](#) (5)
 - [1860-1869](#) (5)
 - [1870-1879](#) (6)
 - [1880-1889](#) (11)
 - [1890-1899](#) (12)
 - [1900-luku](#) (76)
 - [1900-1909](#) (15)
 - [1910-1919](#) (14)
 - [1920-1929](#) (12)
 - [1930-1939](#) (16)
 - [1940-1949](#) (8)
 - [1950-1959](#) (30)
 - [1960-1969](#) (15)
 - [1970-1979](#) (5)
 - [1980-1989](#) (3)
 - [1990-1999](#) (1)
 - 2000-luku

Figure 10: The tree view of MuseumFinland

To balance the scales, and support quick spot searching when the user knows what he is looking for, MuseumFinland includes semantic keyword searching functionality. This functionality is seamlessly integrated with view-based search in the following way: First, the search keywords are matched against category names in the facets as well as text fields in the metadata. Then, a new dynamic view is created in the user interface. This view contains all categories whose name or other defined property value matches the

keyword. Intuitively these categories tell the different interpretations of the keyword, and by selecting one of them a semantically disambiguated choice can be made. This also solves the search problem of finding relevant categories in views that contain thousands of categories. The view in figure 11 includes a keyword search view for the word “nokia”. Matched are, for example, the categories Nokia (the telephone company), Nokia (the place) and Nokia-Mobira (an earlier incarnation of the telephone company). A result set of object hits is also shown. This result set contains all objects contained in any of the categories matched as well as all objects whose metadata directly contains the keyword. The hits are grouped by the categories found.

The screenshot shows a search interface with a search bar on the left containing the text "Käsittehaku:" and a search button labeled "Hae". Below the search bar, there are several facets for the keyword "nokia":

- Hakusana: nokia (poista)**: A list of categories with counts: Valmistaja > ... > [Nokia](#) (14), Valmistuspaikka > ... > [Nokia](#) (32), Käyttöpaikka > ... > [Nokia](#) (4), Valmistaja > ... > [Oy Nokia Ab](#) (1), Valmistaja > ... > [Nokia-Mobira](#) (1), Valmistaja > ... > [Nokian jalkine tehdas Oy](#) (7), Valmistaja > ... > [Nokian Kutomo ja Värjäys Oy](#) (2).
- Esinetyyppi (koko luokittelu) (ryhmittele kohteet)**: [kulkuneuvot ja kuljetusvälineet osineen](#) (1), [koneet ja laitteet](#) (4), [pukineet ja tekstiilit](#) (30), [säilyttimet](#) (2), [leikkikalut](#) (2), [sisustus](#) (1).
- Materiaali (koko luokittelu) (ryhmittele kohteet)**: [materiaalit](#) (37).
- Valmistaja (koko luokittelu) (ryhmittele kohteet)**: [yritykset](#) (34).

On the right side, there is a section titled "Hakuehdot" (Search criteria) showing "Hakusana: nokia (ryhmittele kohteet) (poista)". Below this, it says "Kohteet ryhmiteltyinä hakusanan nokia mu (näytä ilman ryhmittelyä)" and "Valmistaja > [Nokia](#), kohteet 1-4/14 (ryhmittele kohteet)". Two images of toy cars are shown:

-  leluauto:henkilöauto (ECM 3594 47)
-  leluauto:lelukilpa-auto (ECM 3598 1)

At the bottom, there is a link: "Valmistuspaikka > [Nokia](#), kohteet 1-4/32 (ryhmittele kohteet)".

Figure 11: Entering keywords creates a dynamic facet of matching categories


At any point during the view-based search the user can select any hit found by clicking on its image. This moves the user interface into the individual item view, and a mode of browsing the results complementary to view-based search. The individual item view is shown in figure 12. The example depicts a special part, a distaff (“rukinlapa” in Finnish) used in a spinning wheel. On top, there are links to navigate directly in the groups and results of the current query. These allow the user to look at individual results of their query in detail without having to return to the main search view at each interval. For a portal more oriented toward a single search result set, this view should probably occupy more space and resemble the result grouping section of the main search view. However, here there is a conscious effort to make a clear separation between a search and a browsing phase in the portal. This is done in order to encourage the user to use the other means of browsing the portal, detailed soon below.

Below the query navigation links in figure 12, on the left and center, are the detailed metadata about the item stored in the database. In MuseumFinland, this includes the picture or pictures of the item, as well as the content of the original database fields as

MuseoSuomi
- Suomen museot semanttisessa webissä -

Uusi haku | Takaisin hakusivulle | Ohjeet | Tietoa ohjelmasta | MuseoSuomi-palautte
Espoo (180) << | **Bemböle** (14) | >> Järvenpää (9)
(*) **rukinlapa** | (*) Jämsisvuolin

rukinlapa



Valmistuspaikka: Suomi
Valmistusaika: 1793
Käyttöpaikka: Suomi,Bemböle,Espoo,Suomi,Vanhakartano,Espoo,Suomi
Asiasana: KEHRUU, KORISTEVEISTO, PUUMERKKI, VUOSILUKU
Museokokoelma: Museokokoelma
Vastuumuseo: Espoon kaupunginmuseo
Asiasanasto: Espoon kaupunginmuseon sanasto
Esineen numero: ECM:100.1
ID: 1001

Esinetyyppi:

- työvälineet (298) > tekstiikkasivovälineet (219) > kehruun ja langanväristyksen työvälineet (63) > kehruvälineet (59) > kuontalompittimet (3) > **rukinlapa** (1)

Valmistuspaikka:

- Europa (2541) > **Suomi** (2239)

Valmistusaika:

- ajakaudet (3024) > historiallinen aika (3023) > **uusi aika** (3013)
- vuosajat (3012) > **1700-luku** (123)

Käyttöpaikka:

- Europa (2232) > **Suomi** (2227)
- Europa (2232) > **Suomi** (2227) > Etelä-Suomen lääni (1999) > Uusimaa-Nyland (670) > Espoo (512)
- Europa (2232) > **Suomi** (2227) > Etelä-Suomen lääni (1999) > Uusimaa-Nyland (670) > Espoo (512) > **Bemböle** (14)

Käyttötilanne:

- valmistustekniikat (1587) > tekninen työ (39) > veisto (32) > **koristeveisto** (8)
- valmistustekniikat (1587) > tekstiilityö (886) > kututyö (74) > **kehruu** (64)

Kokoelma:

- Espoon kaupunginmuseon kokoelmat (1190) > **Museokokoelma** (1129)

Sama käyttöpaikka

Bemböle:

- jämsivuolin
- opetusväline peli
- opetusväline peli
- opetusväline peli
- opetusväline peli
- opetusväline peli

Espoo:

- kuvakirja kuvakirja, kangasta
- lehdet: lapen lyhythahman lehti
- neuletaki naisen neuletakki
- hartavaate naisen pitsinen hartavaate
- puuvu yläosa, iakkunaisen puvun yläosa

Suomi:

- ruokahina ruokahina, damasti
- kaitalaina kaitalaina, etupistokirjontaa
- pöytäalaina pitkulina, kirjoittu
- pöytäalaina ristipistokirjontainen pöytäalaina
- kaitalaina batistilina, kirjoittu

Esineeseen liittyvään paikkaan liittyviä muinaismuistoja

Espoo:

- Röykkiöt
- Puolustusvarustukset
- Röykkiöt
- Röykkiöt

Samaan aiheeseen liittyviä esineitä

ajan_kasitteet:

- hetvetoimi
- arkkuväsaatarkku
- takki vanuuste
- veistos pienoisveistos
- perukartturukka

kehruu:

- iakkara kehrunjakkara
- rullatuokirullateline
- puolalankapuola
- puolalankarulla
- loukku pellavaloukku

Figure 12: The item view of MuseumFinland

extracted from the museum databases. At the bottom center, the views are again shown, this time in an inverted form, showing all the hierarchy paths to the current item. Clicking on any category here starts a new search for items referring to just that particular category. The idea is that once a user has found an item interesting in some regard in the virtual museum exhibition, they can easily find others like it in that same regard.

This loopback to the search view is however not the only way in which the portal supports browsing based on an interesting item as a starting point. On the right of the item view there are a collection of semantic links coupling other items directly to the one currently viewed. These allow for lateral direct browsing between the items in the portal database as a complementary means of navigation. The idea here is that the view-based search can also be seen only as the starting point for finding one interesting item. The rest of the user experience can consist of “wandering the museum halls” from an object to another related one.

The semantic browsing component of the view is organized as follows: First, a heading is shown describing the rule linking the items together. Then, a subheading shows a

semantic property or properties of the current item that are shared with other items in the collection. These items with common elements are then shown as the actual links.

While most of the link groups are based on the same categorization used in the view-based search, such as “Common location of use” (“Sama käyttöpaikka”), some rules go beyond the view definitions to capture other complex associations between the items. For example in figure 12, under the heading “Items related to the same topic” (“Samaan aiheeseen liittyviä esineitä”), besides other items related to the category “Spinning” (“kehruu”), there are other items related to “Concepts of time” (“ajan_kasitteet”). This is possible despite the fact that the views do not contain any “Concepts of time” category. In the underlying metadata RDF graph it has been annotated that the distaff has a year carved into it, and thus it can be found in the rule doing the semantic linking.

Comparing the interaction patterns of the MuseumFinland virtual exhibition with the physical experience of visiting a museum provides further perspective on the interface. The view-based search can be equated with choosing or building a physical museum exhibition dynamically by selecting items dealing with a certain topic or a combination of topics. The semantic browsing from item to item and item group to item group can be equated with wandering between the exhibits in the exhibition selected. However, one is not limited to a particular way of ordering the items as in a physical space, but can change axis at will. Thus, MuseumFinland provides an experience that is to some extent similar to actually visiting the museum, an experience usually considered to have positive connotations, but hard to achieve on the web.

5.2 View-Based Search in Your Pocket: MuseumFinland Mobile

A goal of OntoViews, the software framework underlying MuseumFinland was to facilitate multi-channel publication. This means that the same content and services can easily be shown to end users in different ways and on different platforms. To demonstrate this, another user interface for MuseumFinland to be used by WAP 2.0 (XHTML/MP) compatible devices was created⁵.

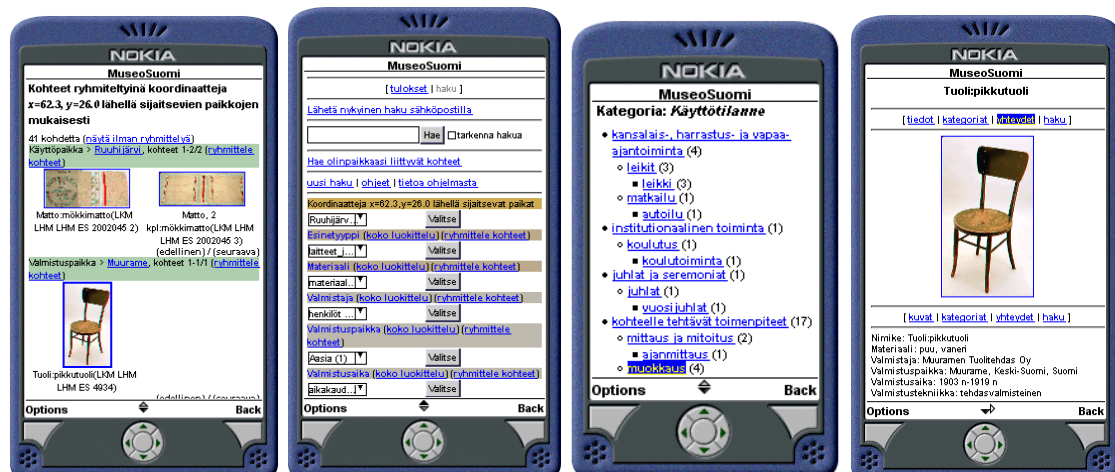
There are many challenges to creating an efficient view-based search interface on a mobile device, where screen space is precious and navigation needs to be quick and effortless. The limited screen size makes it hard to give useful overviews on the items from multiple

⁵The interface responds at the same address <http://www.museosuomi.fi/>. The choice of which interface to show is based on identifying the platform of the client. This, in turn, is done based on the user agent string the client supplies to the server.

viewpoints, and for the MuseumFinland interface, the multitude of steps in drawing up a query can be troublesome.

There are, however, some valuable features in the MuseumFinland interface for mobile use: First, empty results can be completely cut out, which is a useful feature in an environment where navigation is cumbersome and data transfer latencies and costs are still often high. Second, selecting from predefined categories removes the need for typing keywords on mobile keypads. Third, eliminating infeasible choices makes it possible to use the small screen size more efficiently for displaying relevant information. Also, the main problems appear only when starting a search. Once underway, the iterative constraining and semantic browsing functionality inherent in the system provides a simple and effective navigation method in a mobile environment. Thus, in designing the mobile interface, two goals were set: First, to minimize the screen space taken by the views, and second, to provide alternate starting points for browsing, by bootstrapping the search in some way.

In general, the mobile interface repeats all the functionality of the PC interface, but in a layout more suitable to the limited screen space of mobile devices. Figure 13 shows examples of the mobile versions of all the main page types of MuseumFinland.



(a) Results on the main search page

(b) Facets on the main search page

(c) Tree view page

(d) Item page

Figure 13: MuseumFinland Mobile interface

In the mobile interface, current search results are shown first up front noting the current search parameters for easy reference and dismissal. This is depicted in figure 13(a). Below the results, the search facets are shown, depicted in figure 13(b). In the mobile user interface selectable subcategories are not shown as grouped links as in the PC interface, but as drop-down lists. In most mobile browsers, such lists replace the whole view when

selection is started in them. This minimizes screen space use while browsing the facets, but maximizes usability when selecting subcategories from them. In-page links are provided for quick navigation between search results and the search form. The familiar tree view, shown in figure 13(c), provides an alternate way of selecting constraints and viewing the results, agreeable with the limits of a mobile device.

The item page depicted in figure 13(d) is organized in a similar fashion to the browser version shown in figure 12, showing first the item name, images, metadata, annotations, semantic recommendations, and finally navigation in the search result. Again, there are in-page links for jumping quickly between the different parts of the page.

To provide a quick starting point for search and browsing, two new functionalities were created. First, the geographical location of the mobile device can be used as a constraint, realized in the interface in the same manner as the concept-based keyword search. Any entries in the Location ontology with coordinates near the current location of the mobile user are shown in a dynamic facet, as well as all data objects made or used in any of these locations. In addition, any objects directly annotated with geographical coordinates near the mobile user are shown grouped as normal. In the case of MuseumFinland, these are ancient burial sites and other historical nature targets that were included in the semantic database. This geolocation feature gives the user a one-click path to items of likely immediate interest. The results shown in figures 13(a) and 13(b) are actually from such a geographical search, initiated somewhere near Ruuhijärvi, Finland. As a second starting point for locating possibly interesting items, the interface displays on the front page the last items viewed by any portal user, a functionality that has been transferred also to the main web interface.

Finally, because navigation and search with mobile devices is still bound to be more tedious than in a PC environment, the mobile interface of MuseumFinland allows any search state to be “bookmarked”. An URL containing the current state is sent by e-mail, and can be inspected later in more detail by using the more convenient PC interface.

While this mobile user interface of MuseumFinland should be considered a preliminary work, coupling the functionalities presented with the mobile interface considerations described above suggest that the view-based paradigm is at least workable in the difficult domain of mobile interfaces.

5.3 A View-Based Search Interface for Efficient Search: Veturi

Spot searching, most often currently realized through keyword searching, provides the user with a fast way to reach their goal. It requires that the user knows what they are looking for, and additionally knows how to describe it in the terms the search engine requires. Yellow pages directories is a domain where one can often expect users to know what they are looking for. There are no guarantees however that the user can formulate their queries accordingly. In the “Intelligent Web Services” (IWebS)⁶ project, the view-based yellow pages search portal Veturi⁷ [MVL⁺05] was created to address this search problem.

The portal contains some 220,000 real-world services from both the public and private sectors. The contents of the portal were again drawn from multiple databases to fully demonstrate the usefulness of the semantic technologies. The databases used came from the Finnish yellow pages service provider Fonecta⁸ and the National Research and Development Centre for Welfare and Health (Stakes)⁹. Here too, the database contents were semantically enriched, resulting in an annotation backed by a compound service ontology. This ontology, in turn, was built on top of generally available ontologies and classifications, like the Suggested Upper Merged Ontology (SUMO) [NP01, PNL02], the Finnish Standard Industrial Classification (TOL) [Sta02] and the United Nations Classification of Individual Consumption According to Purpose (COICOP) [Uni99].

The idea behind the annotation schema created is to partition a service description into its constituent parts, to cover as many of the distinct aspects of the service as possible. In this way, a user can locate the service she needs based on the aspects most natural or familiar to her, be it the location, producer or actual change process involved. In the schema, a service is represented in terms of events and the roles related to them, such as “patient”, “instrument”, “locality” and “time”. Further, the services can be divided into operational subevents to form processes. As an example, figure 14 depicts the most important ontological structures used in representing a “Rock removing” service. The service is linked to a “consumer” and a “producer”, as well as the subevents of “Destruction” and “Transportation”, with “Rock” and “Gravel” as “patient”, respectively.

Based on this schema, five views were chosen to be projected to the user interface, again selected to provide the user with enough service aspects to approach the problem from.

⁶<http://www.seco.tkk.fi/projects/iwebs/>

⁷Available at <http://demo.seco.tkk.fi/veturi/>

⁸<http://www.fonecta.fi/>

⁹<http://www.stakes.fi/>

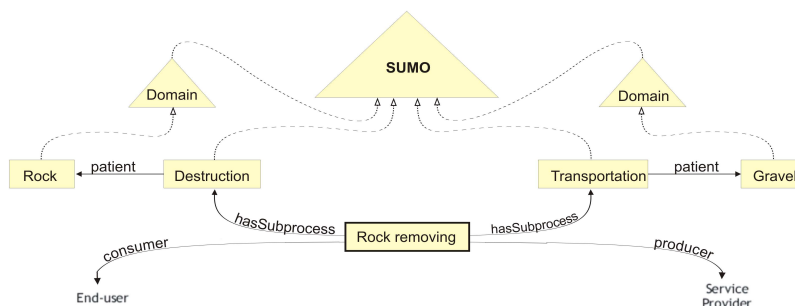


Figure 14: Core ontological structures of the Veturi system

The views and the ontologies underlying them are described in table 3.

Facet view	Underlying ontology
Consumer (Kuluttaja)	Actor
Producer (Tuottaja)	Actor
Patient (Mitä?)	SUMO Object sub-branch + COICOP + Actor
Process (Prosessi)	SUMO Process sub-branch
Location (Paikka)	Location

Table 3: The five view facets in the Veturi portal with Finnish translations, and the domain ontologies they are based on.

To bring the expression power of the service annotation aspects to the user in a quick spot search, the user interface of Veturi is based on on-the-fly semantic autocompletion [HM05] of keywords into categories, made possible by AJAX¹⁰ techniques. This user interaction pattern tightly integrates keyword searching with the specificity, semantic disambiguation and context visualization power of view-based searching.

Figure 15 depicts the search interface of the Veturi portal. The five view-facets used in the portal are located at the top, initially marked only by their name and an empty keyword box. Typing search terms in a box immediately opens the facet to show matching categories, as shown in figure 16. The results view below the facets is also dynamically updated to show relevant hits, defined by any active search constraints in other facets and a union of all the categories in the current facet matched by the keyword. By providing immediate visual feedback to the typed keywords, the common problems of users typing ineffectual keywords and receiving empty result sets can be largely eliminated. On the other hand, if there is a need for more specificity or an alternate selection, a single category

¹⁰Asynchronous JavaScript and the XMLHttpRequest object, which allow for making HTTP calls to the server in the background while viewing a page. See e.g. <http://en.wikipedia.org/wiki/AJAX>.

can be selected from the facet. After such a selection, the facet again closes, showing only the newly selected constraint, with the results view and other facets updating accordingly. As a suitable default for the services domain, the results are sorted into groups based on the location the service is offered in, unless another grouping is specifically requested by the user.

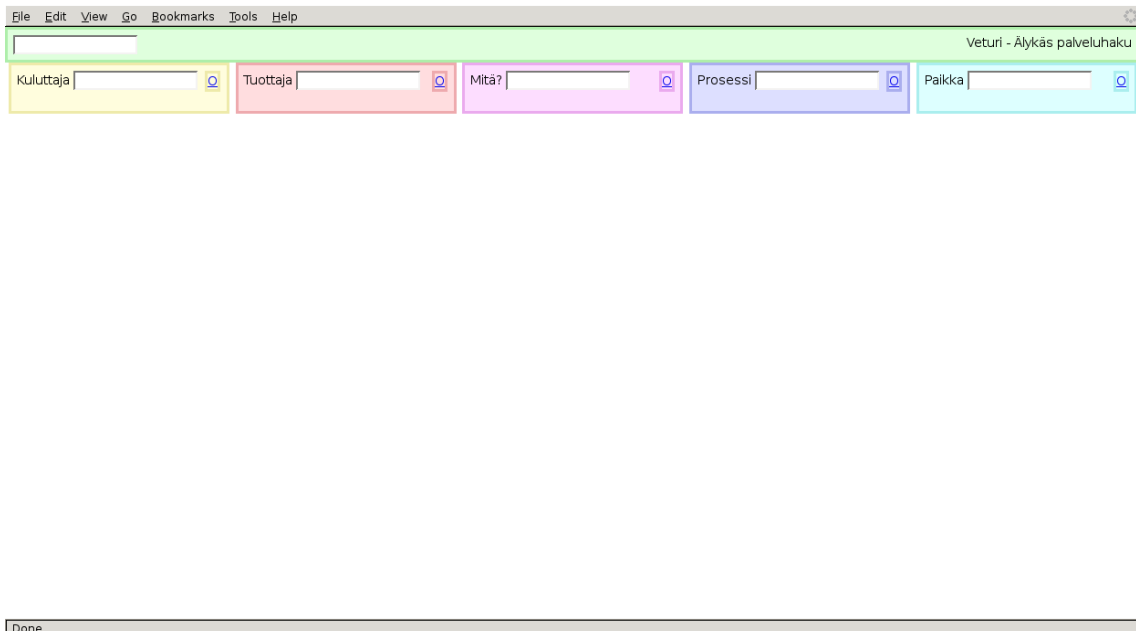


Figure 15: The Veturi user interface in its initial state

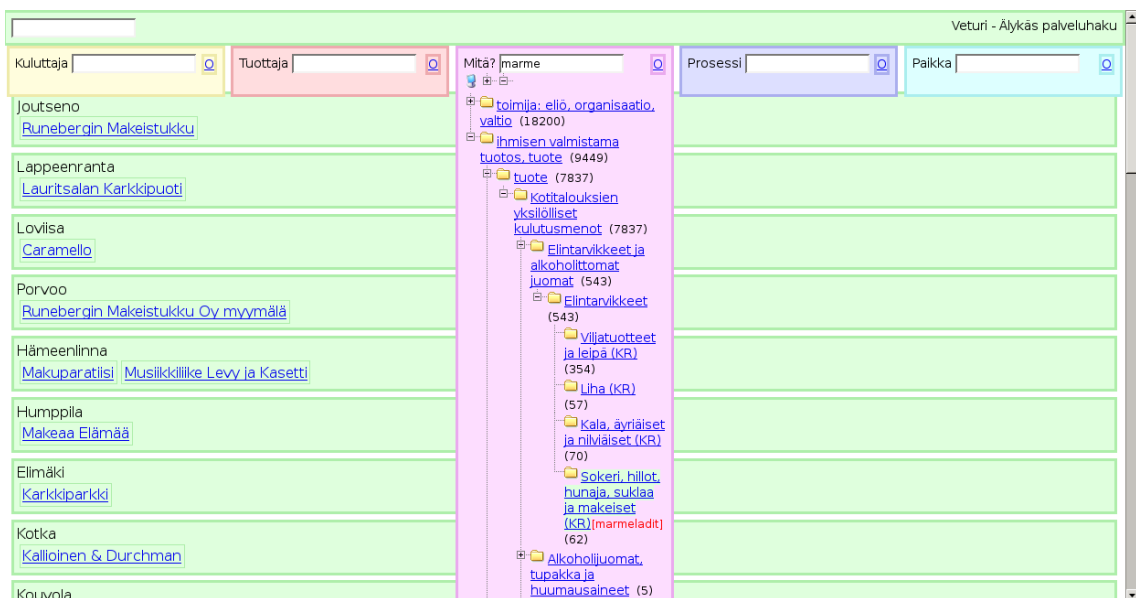


Figure 16: The Veturi user interface after typing “marme” in the Target view

The example search depicted in figure 16 shows a user trying to find out where he can buy marmalade. In the view “Patient” (“Mitä?”), the user has typed in the word ‘marme’. While the annotation ontology used does not contain marmalade directly, the concept category “Sugar, jams, honey, chocolate and sweets” (“Sokeri, hillot, hunaja, suklaa ja makeiset (KR)”) is matched. This is because a descriptive text resource linked to the category in the underlying ontology contains a textual reference to marmalade. In this manner, the search makes use of fuzzy non-ontological knowledge in bootstrapping the semantically firm view-based search. This way, existing textual material can be used to augment incomplete ontologies to at least return some hits for concepts that have not yet been added to the ontology.

Figure 17 shows a continuation of the marmalade search. The search query entered in the view “Process” (“Prosessi”) reveals another feature of the portal: multilanguage support. Typing in the word “buy” matches the correct “business transaction” (“liiketoimi, liiketapahtuma”), even though the word for “buy” in Finnish would be “ostaa”. Here, the benefits of the multiple language support inherent in the RDF data model are simply brought right to the user.

The implementation also supports the T9-type ambiguous numerical input method [DC00, HMNS03] common in mobile phone text input environments. Here, the user needs only to press the number key corresponding to a group of letters on her phone, and the search matches any keyword arising from combining these letter groups. For example, inputting “393” on a Nokia handset would match all possible permutations of “[def][wxyz][def]”, such as “eye”, “dye” and “ewe”.

These keyword search extensions were implemented to demonstrate how the core semantic autocompletion interface can easily be combined with other advances in predictive text autocompletion, because the ontological navigation happens separately after string matching, similarly to what is described in [LTS03].

The Veturi interface has also other benefits. First, the matched categories are shown directly in their hierarchical contexts. This allows for quick evaluation of the relevance of the hits, as well as reveals close misses. For example, a keyword can match a subcategory of a more suitable one, as in when the common-language word “vitamin” has been given, but the whole category of dietary supplements was meant. As a side effect of viewing the trees, the user is also guided on the content of the collection and how it is indexed in the system. The trees can also be opened and navigated freely without using keywords for an alternate form of navigation and familiarization with the indexing concepts and facets.

The user is guided in framing his query by focusing the views on clearly identifiable

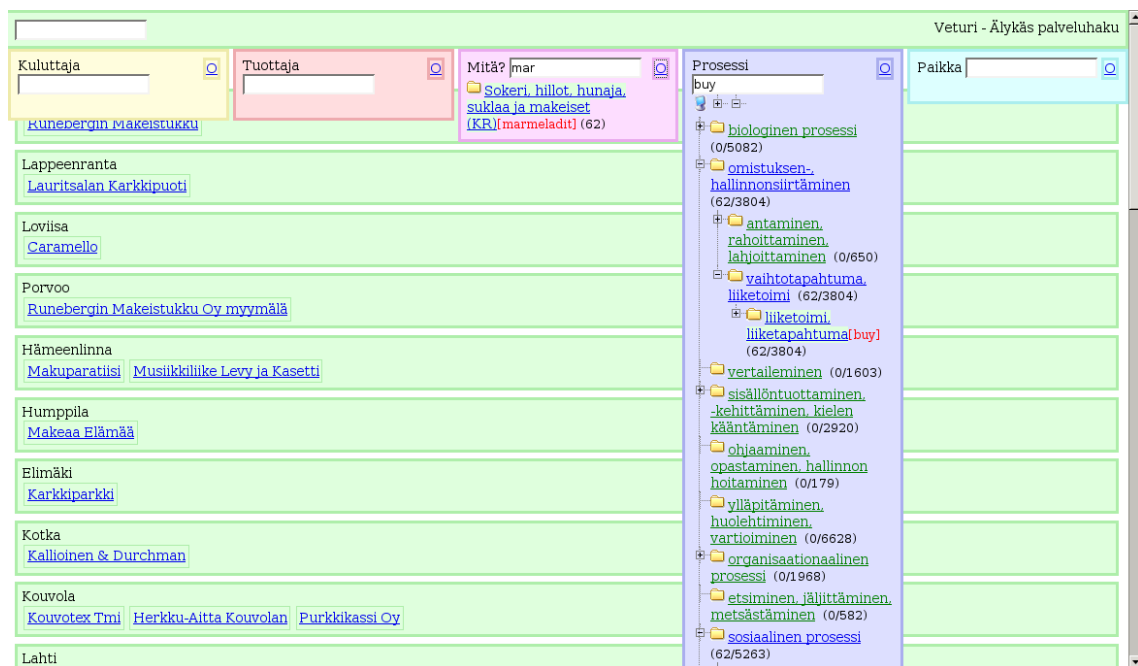


Figure 17: Multilanguage support in the Veturi user interface

distinct qualities of the service. For users more familiar with the portal and its service description model, a globally effective keyword search box for quick, undifferentiated searches is provided in the upper left corner of the interface. Because the contents of the views seldom overlap in the service model used, simply typing the service need in text in the global keyword box usually brings accurate results. If disambiguation is needed, it can quickly be done through selections in the facets. Figure 18 shows the marmalade search done this way.

On selecting an individual service from the results, the user is taken to an item page similar to the one in MuseumFinland, with lateral links to other services in the collection. Here, however, the services are linked using more specific rules, so for example the item page for a hotel shows nearby restaurants and nightclubs, and the item page for a car repair service contains links to nearby taxi companies.

In summary, the Veturi interface provides a powerful tight coupling between the keyword and categorization approaches to service discovery. While both of these approaches have been used in yellow pages interfaces in the past, they have usually been separate. The fact that the Veturi search can be started, and usually also completed simply by typing in keywords provides the users with a familiar entry point to the system. Still, the semantic firmness inherent in the categories is transferred into a sense of security for the user. Users more familiar with a category-based approach are catered to, too, with the added benefit

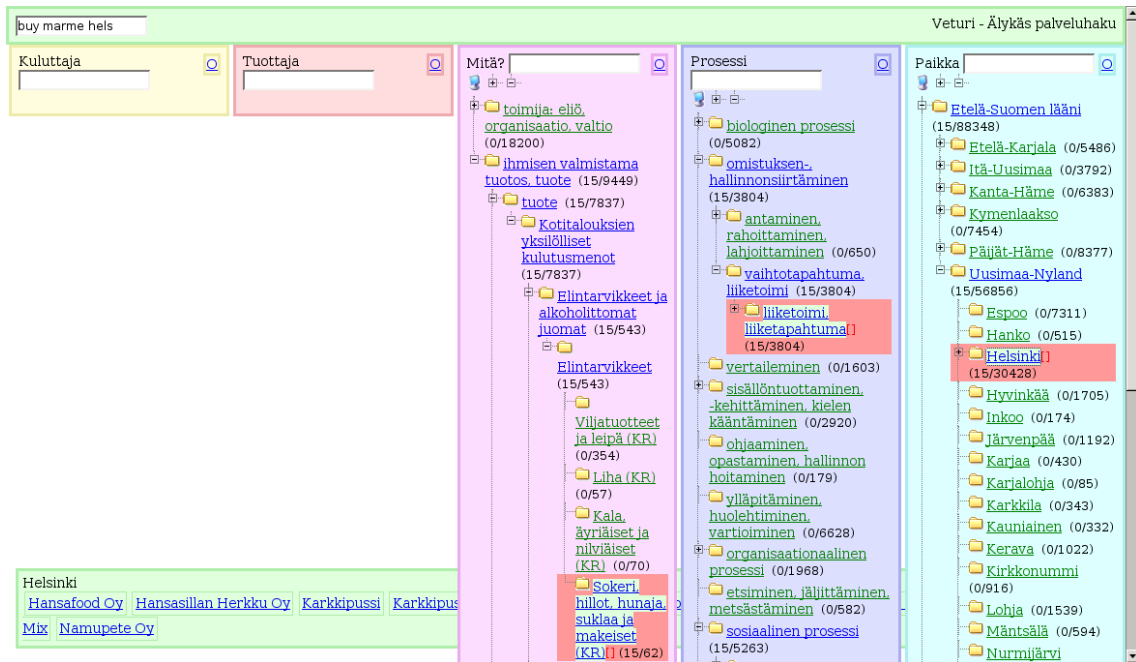


Figure 18: Using the global keyword search box to locate shops selling marmalade in Helsinki

of having multiple viewpoints to choose from, in contrast to the single categorization approaches commonly in use.

5.4 An Alternate Search/Browsing Interface: Orava

The Orava¹¹ portal was originally created by a group of students¹² as a software engineering project at the University of Helsinki Department of Computer Science. The intent was to test how usable the OntoViews portal creation tool was for outside users trying to create a new semantic portal with it. The user interface the group created is an attempt to trim down and reorganize the MuseumFinland interface for a specific kind of data. This was done to make the interface easier to use for new users, overwhelmed by the many possibilities offered by the original interface. The material indexed here consists of some 2200 educational video and audio clips from the Finnish Broadcasting Company YLE, with annotations based on semantically enriching the Learning Object Metadata (LOM) metadata already available for the clips. The interface contains six facets projected from the enriched data: “Teaching subject” (“Oppiaine”), “Theme” (“Teema”), “Target group” (“Kohderyhmä”), “Subject Difficulty” (“Vaativuus”), “Media type” (“Mediatyyppi”) and

¹¹ Available at <http://demo.seco.tkk.fi/orava/>

¹² <http://www.cs.helsinki.fi/group/orava/>

“Program series” (“Ohjelmasarja”). As a particular feature of the data, these view hierarchies are in general quite sparse as well as shallow. Specifically, in each view at each level, there are usually less than ten subcategories, the majority of which are leaf categories. Only a few categories have child categories of their own.

The Orava interface is shown in figure 19. Because of the limited amount of categories in each view, they can be fitted to the left of the interface already at start up without hindering the overview-gaining capabilities provided by the view-based paradigm. The benefit here is giving the interface consistency and leaving the center space for usage instructions. Similarly, because the categories are shallow and this interface is aimed at locating singular objects and not getting overviews of the data, the whole facet views have been cut. To clarify the interface, the choices for grouping results according to the different views have been gathered under a grouping header on the lower right of the screen. This way, they are more likely to be used and understood, compared with their positioning alongside the facets used for selection in the MuseumFinland interface.

The screenshot shows the Orava user interface. On the left, there is a sidebar with navigation menus: **Oppiaine** (Humanistiset tieteet, Kielet, Kuvataide, Liikunta, Luonnontieteet, Matematiikka, Musiikki, Terveystieto, Uskonto, **[X] Yhteiskuntatieteet**, **[X] Psykologia** (90)), **Teema** (Koulutus> (17), Luonto, Työ> (1), Yhteiskunta> (3), Yksilö> (12)), **Kohderyhmä** (**[X] Lukio tai muu toinen aste** (90)), **Vaativuus** (Helppo (30), Keskiavaativa (49), Vaativa (11)), **Mediatyyppi** (Video (89), Aani (1)), and **Ohjelmasarja** (Abitreinit> (19), Avaimia lukemiseen, Ekolokero, Eron stressistä, Kansalaisen ABC, Kesäheinä, Kieliohjelmat, Kiikarissa oma yritys, Kotitallilla, Kätevät, Luontoluuppi). The main content area is titled "Ryhmitelty näkymän Teema mukaan." and shows search results for "mat". The results include abstracts for various psychology topics, such as "Ylioppilastutkinto" (Abitreinit: psykologia, metakognitiot), "Abitreinit: psykologia, tietokonepelit", "Abitreinit: psykologia, väkivaltarikokset", "Abitreinit: psykologia, aggressiivisuuden syöt", "Abitreinit: psykologia, diagnoosin merkitys", "Abitreinit: psykologia, eläinkokeet psykologisessa tutkimuksessa", "Abitreinit: psykologia, isovanhempien merkitys lapselle", "Abitreinit: psykologia, kiusaaminen", "Abitreinit: psykologia, lahjakkuus", "Abitreinit: psykologia, oppiminen ja persoonallisuus", "Abitreinit: psykologia, persoonallisuuden kehitys", and "Abitreinit: psykologia, perusturvallisuus". On the right, there is a search bar with "mat" entered and a "Hae" button. Below the search bar, there is a list of filters: **Oppiaine** (Matematiikka), **Ohjelmasarja** (Kieliohjelmat, Kreikkaa matkailijoille, Portuugalia matkailijoille), **Teema** (Yksilö, Vapaa-aika, Matkailu). Below the filters, there is a section titled "Kohteita:" with a list of links: Etälukio: psykologia, matteesvaikutus, Opinportti: psykologia, psykologiset testit, Etälukio: psykologia, persoonallisuuden testaus. At the bottom right, there is a cartoon illustration of a squirrel holding a red and black target.

Figure 19: The Orava user interface

After receiving the original version of the portal from the student group, some additional

improvements have been made to the interface. First, removing the whole facet view was not without problems. As most categories in the facets are leafs, it was easy for a user scanning the headlines to miss the categories that would yield further subcategory choices. To counteract this, the small notificatory icon “>” was added for categories with subcategories.

A second change was to change the way grouping categories are selected. Originally the grouping categories were selected only from one level in a view tree, as in MuseumFinland. However, because here the amount of categories in a facet as a whole is limited, it was possible to use a flattened version of the whole tree as a basis for grouping the results. This way, more grouping details are made available to the user.

Finally and most importantly, the interface was enhanced with a version of the on-the-fly semantic autocompletion functionality created for Veturi. This functionality can be seen on the right of figure 19, where the user has typed in the string “mat”, and received several matching categories and items. Continuing to type the character “k” would immediately eliminate the category “Mathematics” (“Matematiikka”), and leave only the categories somehow related to “travel” (“matkailu”).

This version of the autocompletion functionality differs from the one in Veturi. While in Veturi the results are shown directly in the general category selection trees, here the matching categories are returned separately, but with their hierarchical context nonetheless. This way, the keyword-based search is more of a separate element from the view-based search. It is used more as a method for bootstrapping the category browsing, than as a tightly integrated component of the whole interaction experience.

On the item page of Orava, first steps were taken to demonstrate how the semantics of the contents can be used to intelligently link multiple OntoViews-based portals together. Several of the categories and ontological concepts in the Orava hierarchies were linked to concepts in MuseumFinland. This allows the user to jump portals during search, for example from a search result in the Orava portal describing the Olympic games direct to items in MuseumFinland related somehow to sports, such as medals.

5.5 Applying View-Based Search for Ontology Browsing: ONKI

The ONKI ontology server [KVH05] is a piece of software developed concurrently with this work at the Semantic Computing Research Group. One of the functionalities of the server is to provide an ontology browsing interface for annotators, where they can find the annotation concepts they need from shared ontologies. As a test and demonstration of the

adaptability of the OntoViews system, a version of the browsing interface was created on top of the platform. Having taken only two days of work, the interface is very preliminary, but still contains the basis of adapting the paradigm to another wholly different task. Here all subsumption hierarchies inherent in the ontology are shown in a combined view, and the user is using the interface not to constrain a query, but to locate relevant concepts in the tree hierarchies themselves.

Figure 20 depicts the OntoViews version of the ONKI user interface. The interface consists of a simple two-plane layout: on the left, the class subsumption tree is depicted, while on the right are shown details for the current class being inspected. In finding concepts the interface relies heavily on the version of the semantic autocomplete functionality that was also included in the Orava interface. Selecting a concept from the keyword search results does two things. First, details of the concept selected are shown on the right. Second, the concept is located in the subsumption tree, and that tree is opened to the proper level.

The screenshot shows the ONKI user interface with a search bar at the top left containing the text 'tuki'. Below the search bar, a list of concepts is displayed, with 'Promootiopäivälliset' selected. The right pane shows details for this concept, including its URI, properties, and a list of related concepts.

Käsitteihaku:

tuki

- [Esiineet](#)
 - [Seppeleenstomismateriaali](#)
 - [Seppeleen tukikaari](#)

Selaa käsitteitä

- [Henkilöt, roolit ja instituutiot](#) +
- [Promootiot ja juhlaistunnot](#) +
- [Paikat](#) +
- [Tapahtumat](#) +
 - [Tapahtumapäivät](#) +
 - [Aktipäivä](#) +
 - [Kutsuvieraiden saapuminen aktisaliin](#)
 - [Juhlajumalanpalvelus](#) +
 - [Promootiopäivälliset](#) +
 - [Promootiopäivällisten puheet](#) +
 - [Promootiopäivällisten runonlausunnat](#) +
 - [Promootiokonsertti](#) +
 - [Promootiokantaatin esitys](#)
 - [Edellisen promootiotoimikunnan tervehdys](#)
 - [Promootioakti](#) +
 - [Kulkue juhlaumalanpalvelukseen](#)
 - [Aktisaliin menevään kulkueeseen järjestäytyminen](#)
 - [Kulkue aktisaliin](#)
 - [Kulkue yliopistolle](#)
 - [Retki ja tanssiaispäivä](#) +
 - [Seppeleensitojaispäivä](#) +
 - [Flooran päivä](#) +
 - [Musiikkiesitykset, puheet ja runonlausunnat](#) +
 - [Tanssitapahtumat](#) +
 - [Kulkuetapahtumat](#) +
 - [Ruokailutapahtumat](#) +
 - [Muut tapahtumat](#) +
 - [Esiineet, tannukset ja kulkuneuvot](#) +
 - [Ohjelmaesitykset, esittäjät, teokset ja tekijät](#) +

Figure 20: The ONKI user interface

On the right, the interface lists the URI of the concept, with all direct properties associated with it. For all the properties whose object is a resource, a link is created that takes the interface straight to that concept. This allows for easily navigating the lateral relationships between the concepts. As the label for the link, the “`rdfs:label`” of the concept referenced

is used if available, falling back to the URI if that fails.

In the navigational plane on the left, only one branch of the subsumption tree is open at each step, with selection closing the current branch and opening another to the level of the selected concept. This is done to keep the amount of concepts on screen manageable. There is, however, an additional feature controlling how the view is visualized. In figure 20, the concept “Promotional dinner” (“Promootiopäivälliset”) has been selected, so the subsumption tree shows that concept, its children, siblings, ancestors and ancestors’ siblings. However, also the grandchild concept “Performing the promotional cantata” (“Promootiokantaatin esitys”) is shown. This is based on a rule in the system, that for categories with only a single subcategory, that category is also shown for selection recursively. The rule was created to eliminate redundant tree navigation along long paths with only a single possible further selection.

Taken back to an interface where the categories are used for constraining search items this would be even more useful, particularly from the viewpoint of using the view facets to also provide information on the current result set. Using this algorithm, the user would be able to directly get the most accurate semantic label describing the current selection, instead of having to open the possibly long path down the view tree from the current selection to see it.

5.6 Adapting the Paradigm to Other Data

While the user interfaces described above were created to solve particular tasks in particular domains, the paradigm and underlying implementation architecture actually allow all to be used interchangeably on any data. Besides the data sets listed above, the interfaces have been tested on a couple of additional knowledge bases to further ascertain the adaptability of the paradigm. These include:

- a knowledge base of university promotion events from the Promoottori system [HSV04],
- data from CultureSampo, the successor system to MuseumFinland with a much wider range of different types of cultural content,
- the dmoz.org open directory project¹³ website directory, and
- link library data from the Suomi.fi¹⁴ e-government portal.

¹³<http://www.dmoz.org/>

¹⁴<http://www.suomi.fi/>

Of these, particularly interesting here are the open directory project data and the Suomi.fi data, because they are both originally single hierarchy classifications of links to information items. The case studies concerning these tell of how such data can be converted to view-based search.

In the case of the dmoz.org data, the top level categories in the single hierarchy actually made workable, if not perfect views. Thus, the portal ended with the views Arts, Business, Computers, Games, Health, Home, Kids and Teens, News, Recreation, Reference, Regional, Science, Shopping, Society, Sports and World.

Unfortunately, the same was not true of the singular topic classification of Suomi.fi. Instead, for the semantic version of the Suomi.fi portal¹⁵, new views were built up from scratch to complement the existing view [SH05]. The new views concerned the type of the content, the language of the content, the target group and their status in life, and any specific spatial region where the data was useful.

While in each of these cases it was possible to create workable views for the application, they highlight a possible restriction on the usability of the paradigm. There may be some data where only a single classification is sensible, or any other categorizations produced do not intersect efficiently with it. This may be true for example with homogeneous collections of articles, where any topic hierarchy cannot be efficiently separated into sensible views.

6 The Architecture of OntoViews

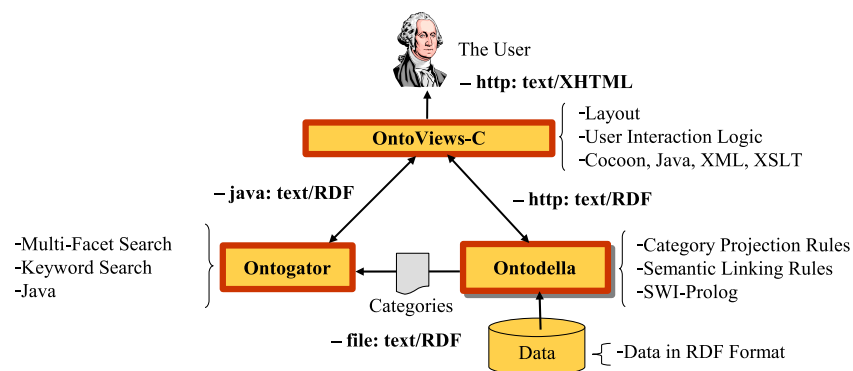
When implementing view-based search on the Semantic Web, some technical design issues surface. This section of this thesis discusses the problems faced and the decisions undertaken while creating OntoViews¹⁶, the view-based search tool underlying all the interfaces described above.

The major design principles underlying the OntoViews tool were to make it: 1) easily adaptable to new underlying domain ontologies, 2) easy to extend and adapt to new user interfaces and interaction patterns, 3) as modular as possible and 4) uphold a clear separation between the major components of the system. In accordance with these guidelines, OntoViews consists of three major components: OntoViews-C, the user interface and interaction controller, Ontogator, the view-based search engine and Ontodella, an SWI-

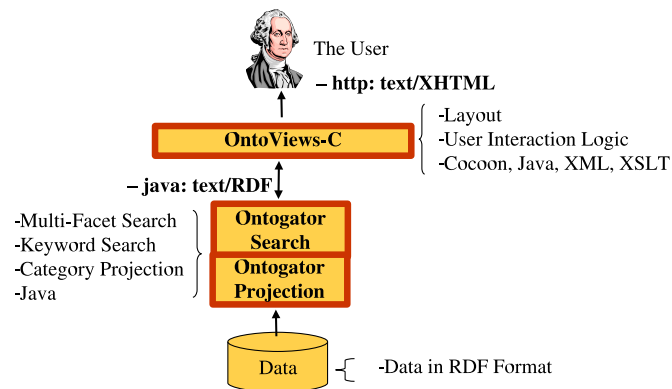
¹⁵Available at <http://demo.seco.tkk.fi/suomifi/>

¹⁶The tool is available for free use, under the MIT open source license at <http://www.seco.tkk.fi/projects/semweb/dist.php>.

Prolog¹⁷-based logic server capable of both projection and item recommendation generation. As a later addition, a separate projection component was also built into Ontogator to get rid of the SWI-Prolog dependency when not doing recommendation-based browsing. Figure 21 shows the main components of the OntoViews architecture, in the two possible ways they can be combined to form a full application: first, using Ontodella to do the projection and recommendation, and second, only relying on Ontogator for both projection and search. The figure also briefly summarizes the interface protocols and formats used between the components. In the following, these three main architectural components will be described in turn, explaining also the interfaces in more detail. Specific questions relating to each component will also be addressed.



(a) OntoViews with semantic linking and projection using Ontodella



(b) OntoViews with projection using Ontogator

Figure 21: The two possible architectural configurations of the OntoViews system

¹⁷<http://www.swi-prolog.org/>

6.1 The Projection and Semantic Linking Engine Ontodella

A crucial part of the adaptability of a view-based search system on the Semantic Web is the flexibility and ease of use of the component responsible for projecting views from the underlying ontology knowledge. In designing the original OntoViews architecture, it was decided to use Prolog-based logic rules as a basis for projection. The power of Prolog allows formulating complex rules when necessary, but the most common case, where projection is based on simple transitive properties, can also be easily and shortly encoded. A similar rationale was also applied to the semantic linking of items with each other, utilized in the semantic browsing part of the user interface.

Because both the projection rules and the semantic linking rules of OntoViews are pure Prolog, the Ontodella logic server component is mostly just a thin wrapper over the core SWI-Prolog engine. Its responsibilities are loading an RDF data model into the engine, organizing projection, listening on an HTTP port for semantic link generation requests and serializing various results produced by the system to RDF/XML for output. At the core of the system are the general formats chosen for the projection and semantic link rules, presented next.

6.1.1 View Projection Rules

In Ontodella, views are projected in a recursive process, starting from the root of the view category tree. A view is defined using the following information: first, the URI for the root resource, second a binary subcategory relation predicate or predicates and third, a binary relation predicate linking the actual information items to the categories in the tree. In addition, each view must have a label. An example view predicate is given below:

```
ontodella_view(
  'http://www.cs.helsinki.fi/seco/ns/2004/03/places#earth',
  place_sub_category,
  place_of_use_item,
  [fi:'Käyttöpaikka', en:'Place of Use'] % the labels
).
```

Here the URI on the second line is the root resource, `place_sub_category` is the name of the subcategory relation predicate and `place_of_use_item` is the item predicate. The label list contains the labels for each supported language, in this case Finnish (fi) and English (en).

The binary subcategory predicate can be based, for example, on a containment property in the following way:

```
place_sub_category( ParentCategory, SubCategory ) :-
  SubCategoryProperty = 'http://www.cs.helsinki.fi/seco/ns/2004/03/places#isContainedBy',
  rdf( SubCategory, SubCategoryProperty, ParentCategory ).
```

The item predicate describes when a given resource item is a member of the given category. For example, `place_of_use_item` in the example above can be described as follows:

```
place_of_use_item( ResourceURI, CategoryURI ) :-
  Relation = 'http://www.cs.helsinki.fi/seco/ns/2004/03/artifacts#usedIn',
  rdf( ResourceURI, Relation, CategoryURI ).
```

Based on these rules, the view trees can be produced by iterating through the predicate `ontodella_view`, and by recursively creating the category hierarchies using the subcategory rules starting from the given root category. At every category, all relevant item resources are attached to the category based on the item rules.

The rules presented before show a simple case. For an example of a more complex rule, consider the subcategory predicate used in Veturi for creating the SUMO [NP01, PNL02]-based hierarchies:

```
% base case, handle categories where we're not told to stop, nor to skip
sumo_sub_category(Source,Target) :-
  Skip = 'http://www.cs.helsinki.fi/group/iwebs/ns/process.owl#skip',
  rdf(Target,'rdfs:subClassOf', Source),
  not(rdf(Target,'sumo_ui:display',Skip)),
  not(sumo_subcategory_not_acceptable(Target)).

% if we're told to skip a category, then do it.
sumo_sub_category(Source,Target) :-
  Skip = 'http://www.cs.helsinki.fi/group/iwebs/ns/process.owl#skip',
  rdf(SubClass,'rdfs:subClassOf', Source),
  rdf(SubClass,'sumo_ui:display', Skip ),
  sumo_sub_category(SubClass,Target).

% don't process MILO categories
sumo_subcategory_not_acceptable(SubClass) :-
  Milo = 'http://reliant.teknowledge.com/DAML/MILO.owl#',
  not(rdf_split_url(Milo,Prop,SubClass)).

% don't process if we're told to stop
sumo_subcategory_not_acceptable(SubClass) :-
  Stop = 'http://www.cs.helsinki.fi/group/iwebs/ns/process.owl#stop',
  rdf( SubClass, 'sumo_ui:display', Stop).

% don't process if someone above us told us to stop
sumo_subcategory_not_acceptable(SubClass) :-
  Stop = 'http://www.cs.helsinki.fi/group/iwebs/ns/process.owl#stop',
  rdf( Y, 'sumo_ui:display', Stop ),
  not( rdf_transitive(SubClass,'rdfs:subClassOf',Y)).
```

Here, while the basis for hierarchy formulation is still the “`rdfs:subClassof`” relationship, complexity arises because it is not used as-is. The class hierarchy of the SUMO ontology is designed mainly to support computerized inference, and is not necessarily intuitive to a human end user. To make the hierarchy less off-putting for a user, two additional rules are used, based on configuration information encoded directly into the RDF data model. First, categories in the middle of the tree that make sense ontologically but not to the user should be skipped, bumping subcategories up one level. Second, sometimes whole subtrees should be eliminated. In addition, in the data model there are also classes of the Mid Level Ontology MILO [NT04] extending the SUMO tree. These are used elsewhere to add textual material to the categories for text-based matching, but are not to be directly processed into the tree.

6.1.2 Semantic Link Rules

Creating lateral semantic links between items in Ontodella is based on rules of the form $p(\textit{SubjectURI}, \textit{TargetURI}, \textit{Explanation})$ that succeed when the resources *SubjectURI* and *TargetURI* are to be linked. The variable *Explanation* is then bound to an explanatory label (string) for the link. In the following, one of the more complex rules in MuseumFinland — linking items related to a common event — is presented as an example:

```
related_by_event( Subject, Target, Explanation ) :-
  ItemTypeProperty =
    'http://www.cs.helsinki.fi/seco/ns/2004/03/artifacts#item_type',
  ItemTypeToEventRelatingProperty =
    'http://www.cs.helsinki.fi/seco/ns/2004/03/mapping#related_to_event',

  % check that both URIs correspond in fact to artifacts
  isArtifact(Subject),
  isArtifact(Target),
  % and are not the same
  Subject \= Target,

  % find all the item types the subject item belongs to
  rdf(Subject, ItemTypeProperty, SubjectItemType),
  rdfs_transitive_subClassOf(SubjectItemType, SubClassOfSubjectItemType),

  % find all the events any of those item types are related to
  rdf(SubClassOfSubjectItemType, ItemTypeToEventRelatingProperty, Event),
  % and events they include or are part of
  (
    rdfs_transitive_subClassOf(Event, SubOrSuperClassOfEvent),
    DescResource=TransitiveSubOrSuperClassOfEvent;
    % or
    rdfs_transitive_subClassOf(SubOrSuperClassOfEvent, Event),
    DescResource=Event;
  ),
),
```



```

% find all item types related to those events
rdf(TargetItemType, ItemTypeToEventRelatingProperty, SubOrSuperClassOfEvent),
% and all their superclasses
rdfs_transitive_subClassOf(SuperClassOfTargetItemType, TargetItemType),

% don't make uninteresting links between items of the same type
SuperClassOfTargetItemType \= SubjectItemType,
not(rdfs_transitive_subClassOf(SuperClassOfTargetItemType, SubjectItemType)),
not(rdfs_transitive_subClassOf(SubjectItemType, SuperClassOfTargetItemType)),

% finally, find all items related to the linked item types
rdf(Target, ItemTypeProperty, SuperClassOfTargetItemType),

list_labels([DescResource], RelLabel),
Explanation=[commonResources(DescResource), label(fi:RelLabel)].

```

The rule goes over several ontologies, first discovering the object types of the objects, then traversing the object type ontology, relating the object types to events, and finally traversing the event ontology looking for common resources. More checks are made to ensure the found target is an artifact and the subject and target are not the same resources. Finally, information about the relation is collected, such as the URI and the label of the common resource, and the result is returned as the link label.

The semantic links shown on the item page of the user interface are produced on-the-fly when the item is selected for viewing in the OntoViews-C control module. The module makes a dynamic HTTP query to Ontodella, and receives, as a result, the link information in an RDF/XML format. This was done to make it possible to add dynamic calculations to the linking formula, for example considering the user profile, recent search behavior or general item popularity between users. While these properties remain as of yet unimplemented in OntoViews, a separate implementation was produced as part of a forerunner system [HSS02], created as a student project¹⁸. For a more thorough evaluation of the Ontodella approach to recommendation, see [Vil06] (in Finnish).

6.2 The Semantic View-Based Search Engine Ontogator

The search engine of OntoViews, Ontogator, is a general-purpose extensible view-based RDF search engine. It was originally built specifically for tree hierarchies, and while further revisions have opened the system a bit to support non-hierarchical categorizations, the interface and optimizations in the system remain specifically tied to tree hierarchy based querying.

As a more technical exposition of a prior version of the engine is already available in

¹⁸<http://www.cs.helsinki.fi/group/wwwmuseo/>

[Saa04] (in Finnish), the description here will pertain to the particularities most relevant to applying the view-based search paradigm on the Semantic Web in general. These are: adaptability to variant domains, interfacing with other semantic components, category identification, extensibility and scalability.

6.2.1 Adaptability to variant domains

On adaptability to variant data, Ontogator relies on the same tree hierarchy projection paradigm explained in chapter 4.2, as provided by Ontodella (chapter 6.1.1), or in a more recent version, Ontogator itself.

The projection functionality for Ontogator was originally implemented to eliminate the SWI-Prolog dependency of OntoViews when not using the semantic link functionality, but another reason was to provide an alternate RDF-based format for describing the view projections. The projection interface was designed to be modular and extensible, so that new projection rule styles and constructs could be created and used interchangeably in the system. This way, even a user not familiar with Prolog could create projections, while also adhering to the Semantic Web ideology by encoding as much configuration data in RDF as possible.

As an example of the configuration format, a snippet from the Veturi portal describing the patient hierarchy, slightly adapted for demonstration purposes, is provided:

```
<ogt:CompoundHierarchy rdf:nodeID="patient">
  <ogt:virtualRoot>
    <ogt:Category>
      <rdfs:label xml:lang="fi">Mitä?</rdfs:label>
      <rdfs:label xml:lang="en">Patient</rdfs:label>
    </ogt:Category>
  </ogt:virtualRoot>
  <ogt:incHierarchy>
    <ogt:HierarchyDefinition>
      <ogt:metaRoot rdf:resource="&object;Object"/>
      <ogt:incProperty rdf:resource="&rdfs;label"/>

      <ogt:subCategoryLink>
        <ogt:ProvaLink rdf:nodeID="coicopSubClasses">
          <ogt:isLeaf>false</ogt:isLeaf>
          <ogt:linkRule>
            rdf(Target,'coicop:hasParent',Source).
          </ogt:linkRule>
        </ogt:ProvaLink>
      </ogt:subCategoryLink>

      <ogt:subCategoryLink rdf:nodeID="sumoSubClasses"/>

      <ogt:itemLink>
        <ogt:RDFFPathLink>
```

```

    <ogt:isLeaf>true</ogt:isLeaf>
    <ogt:linkRule>
      ^sumo:patient^process:subProcess
    </ogt:linkRule>
    </ogt:RDFPathLink>
  </ogt:itemLink>

</ogt:HierarchyDefinition>
</ogt:incHierarchy>
</ogt:CompoundHierarchy>

```

Here, the projection starts by applying the generic component `CompoundHierarchy`, which creates a new hierarchy by including others under a virtual root. Only one hierarchy is included, starting at the “Object” resource, but leaving that out of the projection. The net effect of this is producing a hierarchy starting with the “Object” resource, but switching that root resource to one with labels defined specifically for the view.

Two “`subCategoryLink`” rules for recursively adding subcategories are defined. The first is a simple rule for the COICOP [Uni99] product hierarchy, defined using the Prova¹⁹ language, a Java version of Prolog. The second subcategory rule is not actually defined here, but refers to a Prova definition elsewhere in the RDF document, one almost identical with the Prolog SUMO subcategory rule described in chapter 6.1.1. This possibility for rule reuse is a nice property of the RDF model, though redundant here in the case of Prolog rules.

The third link rule is an “`itemLink`” for linking items, and demonstrates how the system supports multiple concurrent rule formalisms, in this case a simple RDF path format. The backwards path in the example specifies that to locate the service processes associated with a category of objects, one should first locate all processes where the category is specified as the patient type. From there, one can then find the services that contain those subprocesses.

The extensibility of this projection architecture is based on the projection combining only a few well defined component roles to create more complex structures. There are in essence only two types of components in the architecture: those linking individual resources to each other, and those producing resource trees. Based on these roles it is easy to reuse components, for example using the same linkers both for item and subcategory links, or creating a compound hierarchy by including individual hierarchies. Using the RDF data model for configuring the projection further supports this, giving a clear format for expressing these combinatorial structures, and even making it possible to refer to and reuse common component instances.

¹⁹<http://www.prova.ws/>

6.2.2 Interfacing with Other Semantic Components

On the Semantic Web, it is important that the interfaces of programs conform to established standards. To this end, both the queries and results of Ontogator are expressed in RDF. The query interface is defined as an OWL ontology²⁰, and is therefore immediately usable by any application capable of producing either RDF, or XML conforming to the RDF/XML serialization. The interface itself then contains plenty of options to filter, group, cut, annotate and otherwise modify the results returned. These options allow the basic interface to efficiently meet different demands. While a more detailed discussion of these options is outside the scope of this thesis, a few are discussed later, regarding their effects on the scalability of the system.

Because Ontogator mainly works with tree hierarchies inherent in ontologies, it is only natural that also the result of the search engine is expressed as an RDF tree. This tree structure also conforms to a fixed XML-structure. This is done to allow the use of XML tools such as XSLT to process the results. This provides both a fall-back to well established technologies, and allows for the use of tools especially designed to process hierarchical document structures.

6.2.3 Category Identification

Because of the projection, categories in semantic view-based search cannot be identified by the URIs of the original resources. First, the same resources may feature in multiple views, such as when a place is used in both a “Place of Use” and a “Place of Manufacture” view. Second, even inside one view, breaking multiple inheritance may result in cloning resources. Therefore, some method for generating category identifiers is needed.

An important consideration in this is how persistent the created identifiers need to be. In a web application for example, it is often useful for identifiers to stay the same as long as possible, to allow the user to long-term bookmark their search state in their browser. A simple approach for generating persistent category identifiers would start by just concatenating the URIs of categories in the full path from the tree root to the current category to account for multiple inheritance. Then an additional marker would have to be added, for differentiating between the semantic sense by which the actual information items are related to the categories, e.g. “Place of Use” and “Place of Manufacture” again. This will create identifiers resilient to all changes in the underlying ontology knowledge base other than adding or moving categories in the middle of an existing hierarchy. And even in that

²⁰<http://www.cs.helsinki.fi/group/seco/ns/2004/03/ontogator#>

case, good heuristics would be available for relocating lost categories. This will, however, result in very long category identifiers.

If persistence is not critical, many schemes can be applied to generate shorter category identifiers. In Ontogator, a prefix labeling scheme based on subcategory relationships is used: the subcategories of *a* will be identified as *aa*, *ab* and so on. This scheme was selected because it makes finding out the subcategories of a given category very easy, a useful property in result set calculation, described later. The potential problem here is that even if the order in which subcategories are projected is preserved, adding resources to, or removing them from the ontology may result in categories with different identifiers. That is, a category with the identifier *aba* that used to represent e.g. “Finland” could turn out to represent “Norway”, with no means for the system to know about the change. As the original portals created on top of OntoViews were fairly static, this was not judged to be a problem outweighing the benefits.

6.2.4 Extensibility

While the RDF-based query language of Ontogator was designed for querying in projected tree categorizations, it is quite extensible, with the limitation that it currently only supports grouping results into tree categories.

The basic query format is based on two components: an items clause for selecting items for the result set, and a categories clause for selecting a subtree of categories to be used in grouping the results for presentation. This format enables flexibly grouping the results using any category clause, for example organizing items based on a keyword query according to geolocations near the user.

The way both clauses work is based on an extensible set of selectors, components that produce a list of matching resource identifiers based on some criteria particular to them. The current implementation allows searching for view categories using 1) the category identifier, 2) the resource URI of which the category is projected and 3) a keyword, possibly targeted at a specific property value of the category. These category selectors can also be used also to select items. In this case the selector selects all items that relate to the found categories. Items can additionally directly be queried using their own keyword and URI selectors. Different selectors can be combined to form more complex queries using special union and intersection selectors.

Ontogator can be extended by defining and implementing new selectors. This provides a lot of freedom, as the only requirement for a selector is that it produce a list of matching

items. The selector itself can implement its functionality in any way desired. For example, a selector selecting items based on location could act as a mere proxy, relaying the request to a GIS server using the user's current location as a parameter and returning results directly for further processing.

6.2.5 Scalability

The full vision of the Semantic Web requires search engines to be able to process large amounts of data. Therefore, the scalability of the system was an important consideration in the design of Ontogator. With testing on fabricated data, it was deduced that in general, Ontogator performance degrades linearly with respect to both increasing the average number of items related to a category and increasing the amount of categories as a whole, with the amount of items in isolation not having much effect. As for real-world performance, table 4 lists the results of search performance tests done on the major portals presented in this thesis. Because the queries used in the different portals differ in complexity, the results do not scale directly with regard to size, but still approximately conform to the results of the earlier tests.

Portal	Views	Categories	Items	Avg. items / category	Avg. response time
dmoz.org test	21	275,707	2,300,000	8.91	3.50 seconds
Veturi	5	2,637	196,166	128.80	2.70 seconds
MuseumFinland	9	7,637	4,132	5.10	0.22 seconds
SW-Suomi.fi	6	229	152	3.55	0.10 seconds
Orava	5	139	2,142	84.00	0.06 seconds

Table 4: Ontogator performance comparison

Of the performance test results, the ones done on the dmoz.org data provide an obvious comparison point with current web portals, and confirm that this implementation of view-based search is sufficiently scalable for even large amounts of real life data. This scalability in Ontogator has been achieved using a fast memory-resident prefix label indexing scheme, as well as query options restricting result size and necessary processing complexity. These considerations taken are detailed below:

Indexing The tree hierarchy -based search as presented here requires that related to a category, direct subcategories, directly linked items, the transitive closure of linked items

and the path to the tree root can be computed efficiently. The reverse relation of mapping an item to all categories it belongs to also needs to be efficiently calculated.

Ontogator uses custom Java objects (in memory) to model the direct relations of categories and items. All other data related to the categories and items, such as labels or descriptions are retrieved from an associated Jena²¹ RDF model.

Both direct subcategories and directly linked items are recorded in memory for each category to allow for speedy retrieval. A full closure of linked items is not recorded, but calculated at runtime. To do this, Ontogator makes use of a subcategory closure, gathering together all items in all the found subcategories. The subcategory closure itself is acquired efficiently by making use of the prefix labeling scheme used for the categories. After generation, the labels are stored in a lexicographically sorted index, so that the subcategories of any given category are placed immediately after it in the index. This way, any subcategory closure can be listed in $O(\log(n) + n)$ time, by enumerating all categories in the index after the queried resource, until a prefix not matching the current resource is found. The use of prefix labeling also means that the whole path from view root to a given category is directly recorded in its label. Another advantage is that the identifiers are short, and easy to handle using standard Java utility classes.

Result complexity management To decrease result file size as well as result computation complexity, Ontogator provides many options to turn off various result components. If grouping is not wanted, inclusion of categories can be turned off and respectively if items are not desired, their inclusion can be turned off. Turning both off can be used to gain metadata of the query's results, such as number of item or category hits.

The most important of these options, with regards to query efficiency, deals with the hit counts. Turning item hit counting off for categories speeds up the search by a fair amount. Used generally, however, this deprives the tree-views of their important function as categorizations of the data. Therefore, the option makes most sense in pre-queries and background queries, as well as a last effort to increase throughput when dealing with massive amounts of data.

Result breadth management Result breadth options in Ontogator deal with limiting the maximum number of items or categories returned in a single query. They can either be defined globally, or to apply only to specified categories. With options to skip categories

²¹<http://jena.sourceforge.net/>, the leading Java RDF toolkit, developed under an open source license at HP labs

or items, this functionality can also be used for (sub)paging.

In MuseumFinland, a metadata-generating pre-query is used before the actual search query, to optimize the result breadth options used. The query results are used to specify the maximum number of items returned for each shown category — if the result contains only a few categories, more items can be fitted in each category in the user interface.

Result depth management Depending on the nature of the view-based user interface, hierarchies of different depths are needed. Currently Ontogator supports three subhierarchy inclusion options. These are

none No subcategories of found categories are included in the result. This option is used in category keyword queries: only categories directly matching the given keyword will be returned.

direct Direct subcategories of found categories will be included in the result. This option is used to build the basic views in MuseumFinland.

all The whole subhierarchy of found categories will be included in the result. This option is used to show the whole classification page in MuseumFinland, as well as the main view in Veturi, which give the user an overview of how the items are distributed in the hierarchy.

Similar options are available for controlling if and how paths to the selected category from the view root are to be returned.

With result breadth limits, these options can be used to limit the maximum size of the result set. This is especially important in limited bandwidth environments.

6.3 The Interaction and Control Component *OntoViews-C*

The user interface, interaction and control component of *OntoViews*, called *OntoViews-C*, is built on top of Apache Cocoon²². Cocoon is a framework based wholly on XML and the concept of pipelines constructed from different types of components, as illustrated in figure 22. A pipeline always begins with a generator that generates an XML-document. Then follow zero or more transformers that take an XML-document as input and output a document of their own. The pipeline always ends in a serializer that serializes its input

²²<http://cocoon.apache.org/>

into the final result, such as an HTML-page, a PDF-file, or an image. It is also possible for the output of partial pipelines to be combined by aggregation into a single XML-document for further processing. Execution of these pipelines can be tied to different criteria, for example to a combination of the request URI and requesting user-agent.

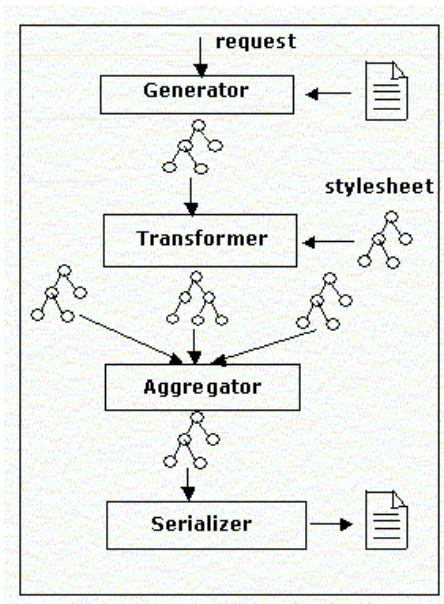


Figure 22: The components of a Cocoon pipeline

In OntoViews-C, all of the intermediate components produce not only XML, but valid RDF/XML. Figure 23 depicts two pipelines of the OntoViews-C system as they appear in MuseumFinland. The pipelines look alike, but result in quite different pages, namely in the search result pages seen in figures 9 and 11 and the item page seen in figure 12. This is due to the modular nature of the pipelines, which makes it possible to split a problem into small units and reuse components.

In OntoViews-C, every pipeline that is tied to user interaction web requests begins with a user state generator that generates an RDF/XML representation of the user's current state. While browsing, the state is encoded wholly in the request URL, which allows for easy bookmarking and also enables the use of multiple responding servers. The user state is then combined with system state information in the form of facet identifiers and query hit counts, and possible user geolocation-based information. This combined information is then transformed into suitable queries for the Ontogator and Ontodella servers, depending on the pipeline.

In the search page pipeline on the left, an Ontogator query returning grouped hits and categories is created. In the item page pipeline on the right, Ontogator is queried for the

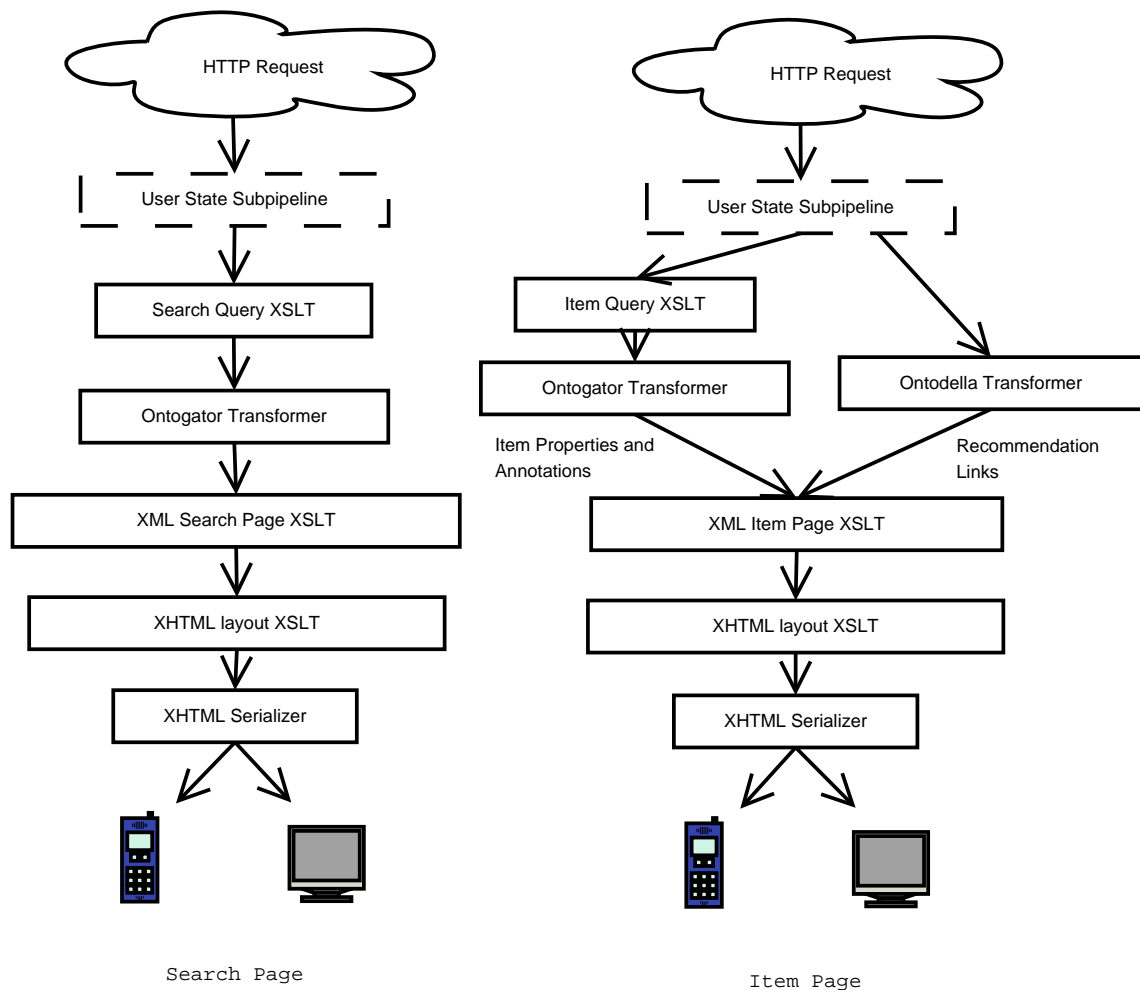


Figure 23: Two Cocoon pipelines used in OntoViews-C

properties and annotations of a specific item and its place in the result set, while Ontodella is queried for the semantic links relating to that item. The Ontogator search engine is encapsulated in a Cocoon transformer, while the Ontodella transformer is actually a generic Web Service transformer that creates an HTTP-query from its input, carries it out, and creates SAX events from the HTTP-response. The RDF/XML responses from the search engines are then given to two consecutive user interface transformers depending on the pipeline, and the device that originated the request. The two different transformers exist to separate application interaction logic with final layout. The first is responsible for creating a logical XML version of the user interface page, and producing all the needed functional links. The second transformer then does the final layout transformation either into XHTML or to another format, which is then finally serialized. In this way, logic and layout are kept separate, and it is for example easy to switch between tree visualizations without even touching the rest of the system. Most of the transformations into queries and

XHTML are implemented with simple XSLT-stylesheets.

7 Discussion

The interfaces described in this thesis confirm the hypothesis that view-based search presents a versatile and powerful paradigm for creating interfaces on the Semantic Web. The paradigm could be applied to solve differing search tasks spanning the full range between the browsing and spot searching strategies. Still, the requirements of the two modes were so different that no one interface could be made to be equally supportive of both. So, while the interfaces created as part of this research do support both modes as much as possible, they differ in which one is prioritized over the other, with the MuseumFinland interface most geared toward browsing, and the Veturi interface most geared toward spot search.

As an indication of the usefulness of the approach, MuseumFinland, the oldest of the portal interfaces, has received several public awards. These include the Semantic Web challenge award 2004 (second place) and the Finnish Prime Minister's commendation for the most technologically innovative application on the web 2004. The portal was also a jury nominated finalist in the Nordic digital excellence in museums awards, in the best Web based / Virtual application category.

Analyzing the interfaces, a number of benefits persist through all of them, explaining the power of the approach:

- Because the collection is visualized along different categorizations, the user is able to immediately familiarize herself with the contents of collection, as well as how it is organized in the database.
- Showing possible constraints in multiple views simultaneously allows the user to start constraining their search with the aspect most natural to them, and continuing from there. This is a particular strength when compared with classifications based on a single hierarchy, such as the ones used in the Yahoo!²³ and Open Directory Project²⁴ directories.
- Because the views show categorizations of the items, and the categories shown are used as search constraints, any information linked to them (such as hit counts) is immediately useful in evaluating further constraining actions.

²³<http://dir.yahoo.com/>

²⁴<http://dmoz.org/>

- Visualizing results from multiple viewpoints is an intuitive, simple way to present how the result set fits into the possibly complex larger context of the domain. It also allows the user to answer questions about sets of items, not just individuals.
- In contrast to keyword searches, the semantic firmness inherent in the categorizations and constraining is transferred into a sense of security in the user of locating all the relevant items.
- Provided that the views intersect efficiently with each other, only a few selections are needed to achieve a wanted narrow result set.

The versatility of the paradigm with regard to client device and environment concerns was confirmed in creating the MuseumFinland Mobile interface. The paradigm also proved to be sufficiently extensible, testified to by the easy and tight integration of keyword- and geolocation-based searching. The Veturi version of keyword-search also makes use of simple ontology navigation to match keywords from ontology entities to categories. Extending the search with lateral semantic browsing functionality could also be done without notable semantic disconnect, based on a natural flow from result set constraining to browsing. Having the view categories double as rules linking the items together provided additional ease.

While the results clearly show that the view-based approach is a good approach to search on the Semantic Web, it still needs to be oriented with respect to other available approaches for both browsing and spot search.

On the browsing side, the most relevant advantage of the view-based search paradigm is the ability to visualize many choices while still preserving their semantic context. A new user will quickly get to know the collection and the way it is organized, as well as be able to locate interesting further choices at each decision point. A drawback is that in the end, the approach is a query constraining method maintaining a result set and query state. This makes it hard to provide view-based browsing as just one equal browsing alternative among many, an often wanted quality in a versatile browsing application. Instead, such as in MuseumFinland, a view-based browsing interface can be used as a starting point for further browsing, familiarizing the user with the collection and allowing them to hone in on an interesting starting point for further exploration.

On the spot search side, all the benefits of the approach apply. Of particular interest, however, is how these qualities compare with those of the other formalisms for creating complex graph patterns discussed in chapter 3.1.3 of the survey section. The paradigm seems more intuitive than the alternatives presented, as in a well-crafted application, most

of the complexity has already been hidden by the designer of the view projection. Based on the same argument, expression power should also not fall notably below the other formalisms. Unfortunately, no formal usability testing between these interfaces has been possible. This is mostly because no stable, obtainable full implementations of the other interfaces exist, and thus any testing would require considerable implementation work and resources not currently available.

Regarding the OntoViews architecture, the implementation has also proved a success. The system has been used to create altogether five different user interfaces. At the same time, the projection functionality has been tested on 8 vastly different data sets. The system was also tested to scale well to large amounts of data using the dmoz.org material.

The Cocoon-based implementation of OntoViews-C is eminently portable, extensible, modifiable, and modular, as seen in the multiple user interfaces designed and presented here. This flexibility is a direct result of designing the application around the Cocoon concepts of transformers and pipelines. This is further confirmed by the fact that a previously tried and abandoned servlet-based approach did not share these qualities.

The modular architecture also allows all the transformer components to be made available for use in other web applications as web services. This can be done as easily as creating a pipeline that generates XML from an HTTP-query and returns its output as XML. In this way, other web applications could make use of the actual RDF data contained in the system, querying the Ontogator and Ontodella servers directly for content data in RDF/XML-format. This also provides a way of distributing the processing in the system to multiple servers. For example, OntoViews-C instances running Ontogator could be installed on multiple servers, and a main OntoViews-C handling user interaction could distribute queries among these servers in a round-robin fashion to balance load.

The use of XSLT in most of the user interface and query transformations makes it simple to carry out changes in layout and functionality. For example, creating the MuseumFinland Mobile interface and the ONKI browser interface both took less than three days of implementation work. A probable explanation for the ease XSLT brings lies in the prevalence of trees in both the queries, query results and user interface visualizations, as XSLT is specifically designed for processing such hierarchically structured documents.

However, in general there are also some problems in using XSLT with RDF/XML. The same RDF triple can be represented in XML in different ways but an XSLT template can be only tied to a specific representation. In the current system, this problem can be avoided because the RDF/XML serialization formats used by each of the subcomponents of the system are known, but in a general web service environment, this could cause

complications. The core search engine components of OntoViews would however be unaffected even in this case, because they handle their input with true RDF semantics.

During the time frame of this research, other implementations of view-based search for the Semantic Web have also surfaced. The Longwell RDF browser²⁵ provides a general view-based search interface for any data. However, it supports only flat, RDF-property-based views. The SWED directory portal [RSC04] is a semantic view-based search portal for environmental organizations and projects, with an interface very similar to MuseumFinland, but lacking semantic keyword search, the whole classification view and semantic browsing functionality. Also, the view hierarchies are not projections from full-fledged ontologies, but are manually crafted using the W3C SKOS [MB05] schema for simple thesauri. The portal does, however, support distributed maintenance of the portal data. The Seamark Navigator²⁶ by Siderean Software, Inc. is a commercial implementation of view-based semantic search. It also, however, only supports simple flat categorizations.

8 Future Work

The interfaces presented in this thesis are only a beginning. With the newly popularized AJAX and equivalent technologies, it is now possible to create much more configurable, dynamic and responsive interfaces also on the web. This chapter applies the experiences and insights learned during the research process toward visioning such a next generation view-based search interface, as well as discusses the architectural changes necessary to facilitate it.

The interface presented here is indented as a general view-based information management tool, an environment for view-based search capable of being configured to provide for both browsing and spot search. Figure 24 contains a user interface mock-up, showing the base on which this versatility is built. The layout of the interface is based on a configurable lattice of non-overlapping window frames, capable of holding any views. This notion comes from a usability-wise acclaimed family of X11 window managers, the most popular of which is Ion²⁷, but of course the frame-based layout itself is familiar also from many every-day applications such as e-mail clients. A frame-based layout also underpins most websites today, so extending from it will give the user a familiar basis for the interface. In support of the paradigm are also the arguments [BK05] of the Haystack developers,

²⁵<http://simile.mit.edu/longwell/>

²⁶<http://siderean.com/products.html>

²⁷<http://iki.fi/tuomov/ion/>

who also chose the customizable frame lattice as the basis for their semantic workspace designer.

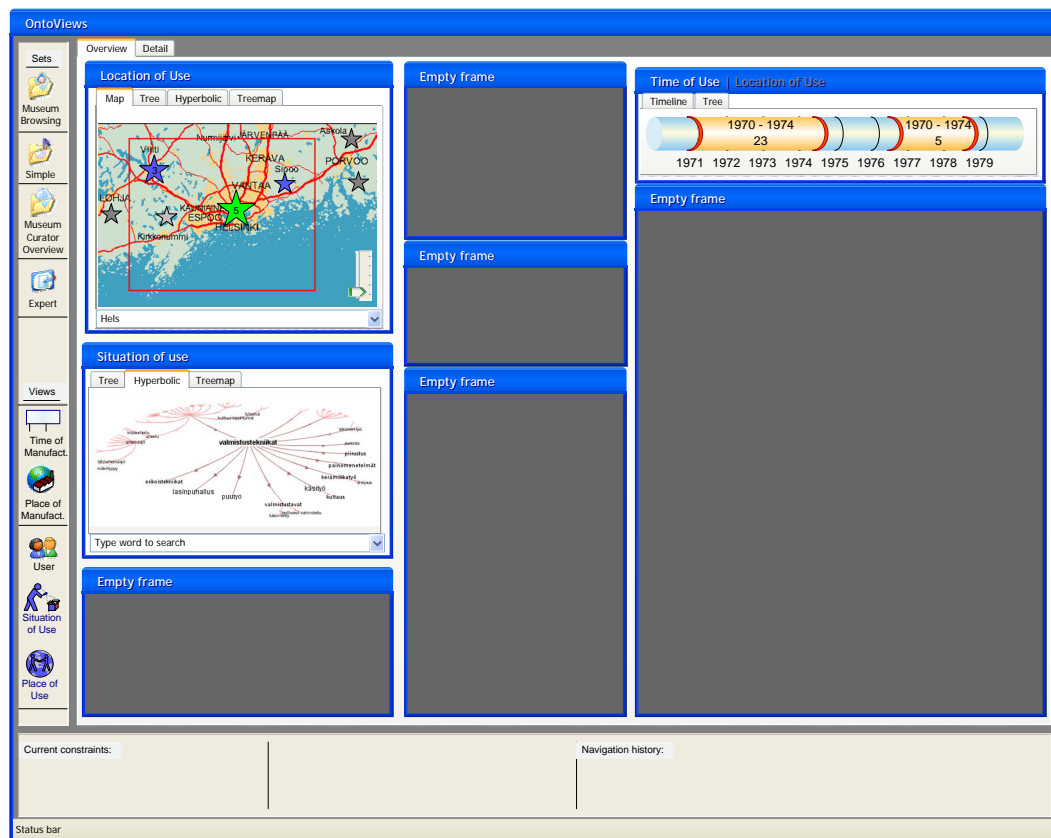


Figure 24: User interface mock-up for the base of a possible next generation view-based search application

An easy way to allow the user to create a frame lattice suitable for their needs is to start with one frame, and then split frames recursively either horizontally or vertically, adjusting dimensions as desired. Once a suitable frame arrangement is arrived at, the act of assigning views to the frames should be as easy as dragging the icon of the view on top of the frame. Each frame could also contain multiple views as alternate tabs, as depicted in the heading of the mock-up frame “Time of use | Location of use”. To make it possible to use views needing a lot of space, a frame could be made to expand to full screen and back by for example double-clicking on its heading.

There should also be a possibility to have several of these layouts with allocated views simultaneously, to counter the fact that a typical view takes up quite much of screen space. Optimally, these view layouts should also take on semantic meaning. As an example, figures 25 and 26 depict two semantically grouped view layouts, that together provide a MuseumFinland-like user interface experience. In the mock-ups, moving between these

layouts is carried out by tabbing. The user interface should also allow these layouts, as well as any views assigned to them to be stored and easily recalled. To this effect, the mock-up contains a “view sets” tab on the left, in addition to the view icons.

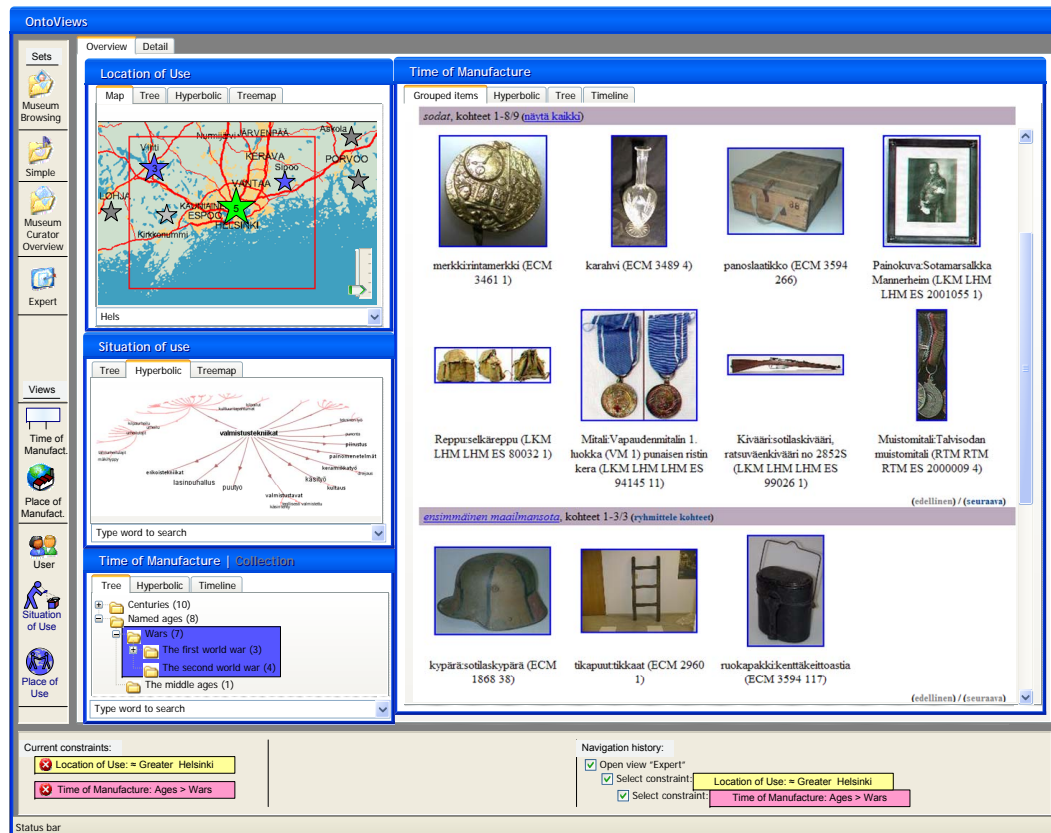


Figure 25: User interface mock-up of a MuseumFinland-like search view layout in a next generation view-based search application

Already evident in the mock-ups so far, but further elaborated in figure 27 is that even if a view is based on a hierarchical categorization, there are other ways of visualizing it than a tree. For example, any tree can be visualized also as a hyperbolic tree (“Situation of Use”) or a treemap (“Material”). Particular views have specific good visualizations available to them, such as using a map for locations (“Location of Use”) or a timeline for events (“Time of Use”). Though, actually, there are still reasons to keep the tree-based visualization as a default. Visualization uniformity across views is a plus, and the trees also take up fairly little space compared with more complex visualizations. Of note is also that even the “Grouping” display is just a view, although one geared much more toward selecting individual items for display than adding further constraints to a query.

Among the views depicted in figure 27 is also a “Complex Ontology Pattern” view, demonstrating how in theory, one could combine to view-based search the other com-

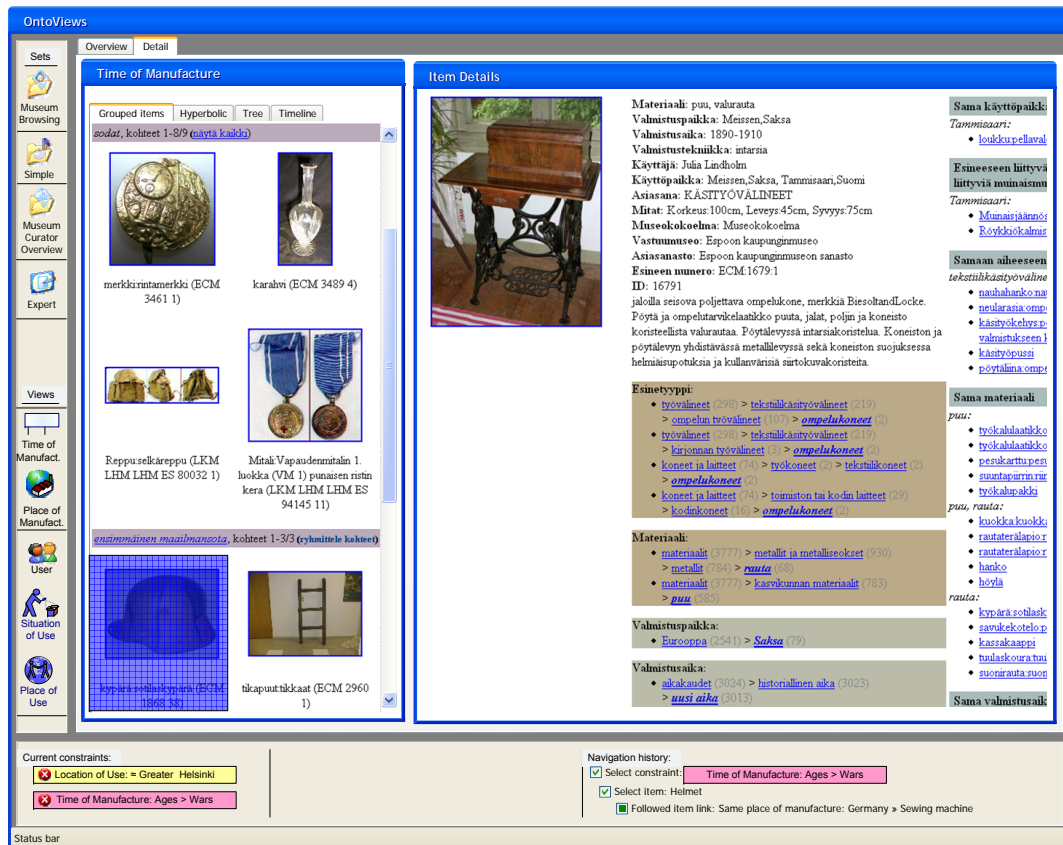


Figure 26: User interface mock-up of a MuseumFinland-like item view layout in a next generation view-based search application

plex graph pattern creation approaches described in chapter 3.1.3 of the survey section. Because there are actually very few requirements for a view in view-based search, it is possible to take one of the complex pattern forming interfaces in as a view. As a usability guideline, though, a view should be able to be used to both constrain, as well as visualize data. In the example, this could be accomplished by adding to the view information on how many individuals of each class type match the pattern. A more serious problem would be that such complex views compromise the semantic simplicity of view-based querying, and introduce redundancy in how different queries can be formulated.

There are also other extensions and usability improvements that could be taken into such a view-based search system. For example, the user could be given quicker and more thorough feedback when considering choices. One way to do this would be that while the user is hovering over possible constraint choices, the interface would dynamically temporarily update to show the results of that choice. Another possible improvement could be an automatic recommendation system, highlighting at each step in the constraining process

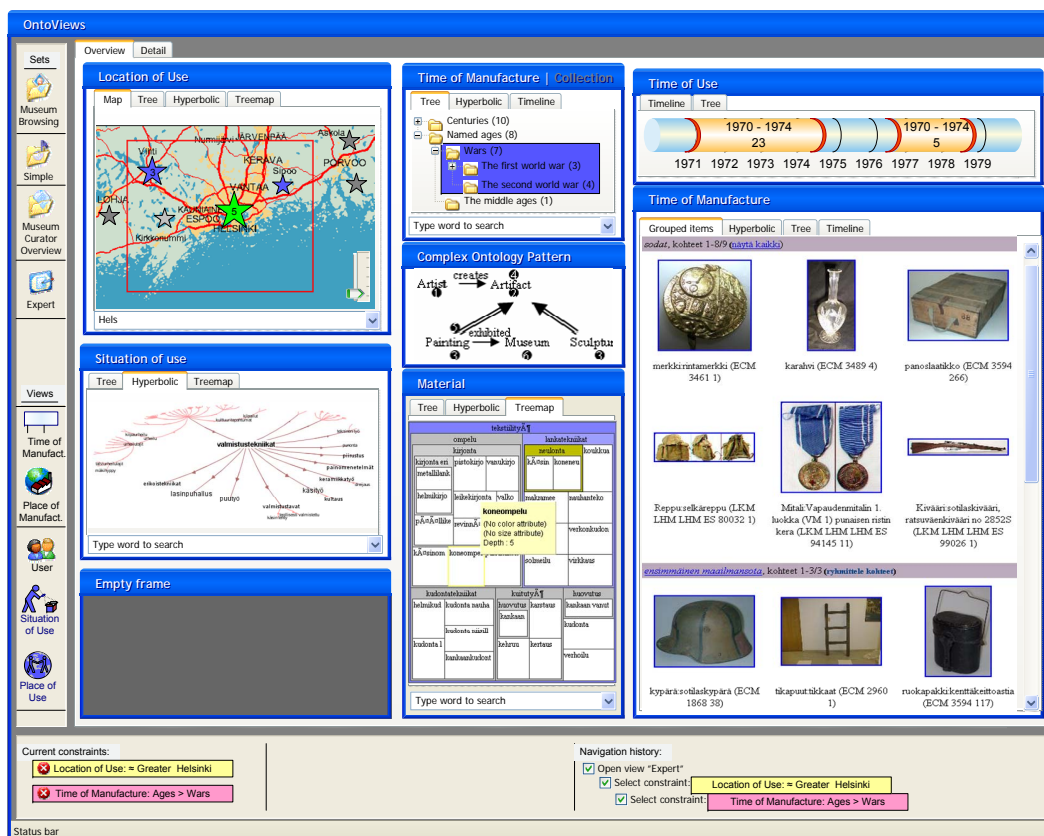


Figure 27: Possible views in a next generation view-based search application

choices and even new views the system considers worthwhile. Such recommendations could be based for example on the result set size resulting from each choice, general user behavior, profiling or any underlying semantic information in the knowledge base. It shouldn't also be hard to move from the logical constraint intersection to a fuzzy or probabilistic one, with the views showing fuzzy membership values. The approaches outlined in chapter 3.2.5 of the survey section would provide an excellent base for this.

With a flexible background architecture, the framework depicted here could form a general semantic view-based window to the Semantic Web. Information to be visualized could be collected from multiple sources, and even the view layouts as well as the view projection rules themselves could be stored as RDF, to be published and shared. This would create a social aspect to the system, and bring it much closer to the vision of the Semantic Web. Then, for example, a view projection made by a user for a certain ontology could be imported by others, making any data on the web annotated using that ontology immediately viewable in the system. To make it easier for users to project views, it would be possible to create a graphical view editor handling most common cases, with only complex projections requiring any significant rules knowledge.

While most of what was depicted here is currently only at the level of ideas, work is already underway at the Semantic Computing Research Group toward implementing these ideas in the context of new view-based search portals. In order to facilitate this, also several parts of the OntoViews architecture need to undergo re-evaluation.

On the controller and general architecture side, the Cocoon architecture and keeping the components of the system as distinct as possible provide flexibility. However, the framework starts to run into problems in interfaces that require tight integration between the server and browser, like the AJAX-powered Veturi interface and the one presented above. For every update of the interface, a whole pipeline must be constructed, and there are no easy facilities for maintaining application state. For example, when navigating tree hierarchies in the Veturi interface, most queries are just opening further branches in a result tree already partially calculated. Currently, however, the whole visible tree needs to be recalculated and returned, because the view-based search is isolated into a separate component.

These requirements also have effect inside the Ontogator search engine. A move is needed from an expectation of monolithic queries and responses to providing all sorts of view-based search related services, tightly integrated with an outside query execution controller that directs the query as the application demands. Another barrier for expansion in Ontogator is its history as a purely tree hierarchy view-based search engine, with grouping into tree categories tightly coupled into the implementation. A solution would be to partition the query processing into two completely separate components: first, a result set generator based on arbitrary selectors and then second, a component for grouping items into arbitrary views. This separation could also further the reuse of the components in tasks other than view-based search.

The use of XSLT in most of the user interface and query transformations makes it easy to adapt the interface appearance and to add new functionality. However, it has also led to some complicated XSLT templates in the more involved areas of user interaction logic, for example, (sub)paging and navigating the search result pages. Therefore, in the next evolution of the system, the interaction logic will probably be pushed back to Java code, with only the layout done using templates.

9 Conclusion

This thesis presented the view-based search paradigm as a viable basis for querying on the Semantic Web, especially coupled with the idea of view projection from ontologies.

Through testing with actual implementations, the paradigm was found to be very flexible and extensible in creating a wide range of interfaces suitable for different tasks in different environments. A design for a general information management tool using the paradigm was also presented.

When other choices are available, the paradigm should especially be considered when:

- there is a need to express complex combinatorial queries intuitively
- there is a need for visualizing result sets, not just individual items
- the content are complex enough that the choice of a constraining viewpoint is useful
- there is use in allowing the user to gently familiarize themselves with the contents and organization of the portal.

Then also, the following drawbacks should be weighed:

- The paradigm is overarching, in the sense that it is hard to integrate other search functionality other than as subservient to the views.
- Other browsing functionality is similarly affected. However, the paradigm can be used to bootstrap browsing.
- Some data may not contain the material needed to produce useful views.

When deciding upon views to be projected, the following considerations apply:

- A good view should both visually organize, as well as be able to be used to constrain the query.
- Views should provide as separate viewpoints as possible to the data. Views with overlapping semantics are possible, but can be confusing.
- For quick operation, the views should intersect efficiently with each other, so that selecting constraints from different viewpoints quickly narrows down the result set.

On the implementation side, OntoViews, a versatile tool for creating semantic portals based on the view-based search paradigm was built. With it, to date eight different semantic portals have been built, with five vastly different user interfaces. The interfaces have been built by varied people, including a student group not directly affiliated with

the research group that created the tool. In all of these cases, the work required to adapt the framework was viewed quick and fairly painless. While new innovations in thinking about view-based search are starting to push the limits of the tool, any future design should be informed by the means by which the modularity, scalability and adaptability of the system was achieved. Particularly:

- Prolog rules provide a flexible base on which to build view projection. However, they are unfamiliar to most users. With thought on the most common projection cases, a simpler, more constrained formalism could be developed to handle the majority of projection, with the added benefit of making a graphical view editor possible. The projection rules of Ontogator provide a starting point for further thinking on the matter.
- For scalable tree hierarchy-based search, an efficient index for calculating a transitive closure of items is needed, and it should be possible to curtail result calculation complexity with options. Also, category identification needs to be sorted out.
- To increase adaptability and component reuse, the old UNIX motto for creating distinct components that do one thing well, but can be connected to perform complex operations continues to apply. On the Semantic Web, it makes sense for the components to both consume and produce RDF as their interface.
- Maintaining a separation between a logical representation of user interface state and the final layout increases flexibility. Using a recursive template-based transformation tool such as XSLT proved a good fit to add even more maneuvering room, particularly for manipulating trees.

Acknowledgments

The author of this thesis wishes to recognize several important contributions to the work presented here, as described in the following. The original idea for combining the view-based search paradigm with the Semantic Web is by professor Eero Hyvönen, who also guided all the research. The chapter on the MuseumFinland user interface bears a heavy influence from his writings.

Except for the extension to support Prova projection rules and limited discussion on design refactoring, the current Ontogator search engine was designed and implemented by Samppa Saarela. He also collaborated in smaller part on the design and implementation of

the first version of OntoViews-C, as well as the first complete version of the MuseumFinland interface. The chapter on Ontogator is based on an unfinished article draft created in collaboration with him.

The Ontodella server infrastructure, as well as the projection and recommendation rule formats are the work of Arttu Valo and Kim Viljanen. The Ontodella chapter is based on a joint publication with the latter.

The Orava user interface was created by a group of students²⁸ as a software engineering project at the University of Helsinki Department of Computer Science, with Teppo Kän-sälä later integrating the semantic keyword autocompletion functionality to the portal.

The application of the OntoViews framework to the Suomi.fi data was the work of Teemu Sidoroff.

References

- ACK04 Athanasis, N., Christophides, V. and Kotzinos, D., Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL). *Proceedings of the Third International Semantic Web Conference*, Nov 2004, pages 486–501.
- AJS⁺04 Airio, E., Järvelin, K., Saatsi, P., Kekäläinen, J. and Suomela, S., CIRI - an ontology-based query interface for text retrieval. *Web Intelligence: Proceedings of the 11th Finnish Artificial Intelligence Conference*, Hyvönen, E., Kauppinen, T., Salminen, M., Viljanen, K. and Ala-Siuru, P., editors, September 2004.
- AS03 Anyanwu, K. and Sheth, A. P., ρ -queries: enabling querying for semantic associations on the semantic web. *Proceedings of the 12th international conference on World Wide Web*, May 2003, pages 690–699.
- BG04 Brickley, D. and Guha, R., editors, *RDF Vocabulary Description Language 1.0: RDF Schema*. World Wide Web Consortium, February 2004. URL <http://www.w3.org/TR/rdf-schema/>. W3C Recommendation.
- BHL99 Bray, T., Hollander, D. and Layman, A., editors, *Namespaces in XML*. World Wide Web Consortium, January 1999. URL <http://www.w3.org/TR/REC-xml-names/>. W3C Recommendation.

²⁸<http://www.cs.helsinki.fi/group/orava/>

- BK05 Bakshi, K. and Karger, D. R., Semantic web applications. *Proceedings of the End User Semantic Web Interaction Workshop (ISWC2005)*, nov 2005.
- BLHL01 Berners-Lee, T., Hendler, J. and Lassila, O., The semantic web. *Scientific American*, 284,5(2001), pages 34–43.
- BRA05 Buscaldi, D., Rosso, P. and Arnal, E. S., A WordNet-based query expansion method for geographical information retrieval. *Working Notes for the CLEF Workshop*, Sep 2005.
- CDM⁺04 Catarci, T., Dongilli, P., Mascio, T. D., Franconi, E., Santucci, G. and Tesaris, S., An ontology based visual tool for query formulation support. *Proceedings of the 16th European Conference on Artificial Intelligence*. IOS Press, Aug 2004, pages 308–312.
- CGPLC⁺03 Corcho, O., Gómez-Pérez, A., López-Cima, A., López-García, V. and del Carmen Suárez-Figueroa, M., ODESeW. automatic generation of knowledge portals for intranets and extranets. *International Semantic Web Conference*, Fensel, D., Sycara, K. P. and Mylopoulos, J., editors, volume 2870 of *Lecture Notes in Computer Science*. Springer, 2003, pages 802–817.
- DC00 Dunlop, M. D. and Crossan, A., Predictive text entry methods for mobile phones. *Personal Technologies*, 4,2(2000).
- EHS⁺03 English, J., Hearst, M., Sinha, R., Swearingen, K. and Lee, K.-P., Flexible search and navigation using faceted metadata. Technical Report, University of Berkeley, School of Information Management and Systems, 2003. Submitted for publication.
- Fel98 Fellbaum, C., editor, *WordNet. An electronic lexical database*. The MIT Press, Cambridge, Massachusetts, May 1998.
- FHH03 Fikes, R., Hayes, P. and Horrocks, I., OWL-QL: A language for deductive query answering on the semantic web. Technical Report, Knowledge Systems Laboratory, Stanford University, Stanford, CA, 2003.
- GMM03 Guha, R., McCool, R. and Miller, E., Semantic search. *WWW '03: Proceedings of the 12th international conference on World Wide Web*. ACM Press, 2003, pages 700–709.

- Hay04 Hayes, P., editor, *RDF Semantics*. World Wide Web Consortium, February 2004. URL <http://www.w3.org/TR/rdf-mt/>. W3C Recommendation.
- Hea00 Hearst, M. A., Next generation web search: Setting our sites. *IEEE Data Engineering Bulletin*, 23,3(2000), pages 38–48.
- HEE⁺02 Hearst, M., Elliott, A., English, J., Sinha, R., Swearingen, K. and Lee, K.-P., Finding the flow in web site search. *CACM*, 45,9(2002), pages 42–49.
- HH00 Heflin, J. and Hendler, J., Searching the web with SHOE. *Artificial Intelligence for Web Search, Papers from the workshop, AAAI 2000*. AAAI Press, 2000, pages 35–40.
- HM03 Hsu, E. I. and McGuinness, D. L., Wine Agent: Semantic web testbed application. *Description Logics*, Calvanese, D., Giacomo, G. D. and Franconi, E., editors, volume 81 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.
- HM05 Hyvönen, E. and Mäkelä, E., Semantic autocompletion. URL <http://www.seco.hut.fi/publications/2005/hyvonen-makela-semantic-autoco> Unpublished, 2005.
- HMNS03 Hasselgren, J., Montnemery, E., Nugues, P. and Svensson, M., HMS: A predictive text entry method using bigrams. *Proceedings of the Workshop on Language Modeling for Text Entry Methods, 10th Conference of the European Chapter of the Association of Computational Linguistics*. Association for Computational Linguistics, 2003, pages 43–49.
- HMS⁺05 Hyvönen, E., Mäkelä, E., Salminen, M., Valo, A., Viljanen, K., Saarela, S., Junnila, M. and Kettula, S., MuseumFinland – Finnish museums on the semantic web. *Journal of Web Semantics*, 3,2(2005), page 25.
- HSJ04 Hyvönen, E., Salminen, M. and Junnila, M., Annotation of heterogeneous database content for the semantic web. *Proceedings of the 4th International Workshop on Knowledge Markup and Semantic Annotation (SemAnnot 2004)*, Nov 2004.
- HSS02 Hyvönen, E., Styrman, A. and Saarela, S., Ontology-based image retrieval. *Towards the semantic web and web services. Proceedings of the XML Finland 2002 conference. Helsinki, Finland*, Hyvönen, E. and Klemettinen, M.,

editors, number 2002-03 in HIIT Publications. Helsinki Institute for Information Technology (HIIT), Helsinki, Finland, 2002, pages 15–27.

- HSV04 Hyvönen, E., Saarela, S. and Viljanen, K., Application of ontology-based techniques to view-based semantic search and browsing. *Proceedings of the First European Semantic Web Symposium, May 10-12, Heraklion, Greece*. Springer-Verlag, Berlin, 2004.
- KBH⁺05 Karger, D. R., Bakshi, K., Huynh, D., Quan, D. and Sinha, V., Haystack: A general-purpose information management tool for end users based on semistructured data. *Proceedings of the CIDR Conference, 2005*, pages 13–26.
- KH05 Kauppinen, T. and Hyvönen, E., Modeling and reasoning about changes in ontology time series. In *Ontologies in the Context of Information Systems*, Kishore, R., Ramesh, R. and Sharman, R., editors, Springer-Verlag, Dec 2005. In press.
- KNRK05 Kruse, P. M., Naujoks, A., Roesner, D. and Kunze, M., Clever Search: A WordNet based wrapper for internet search engines. *Proceedings of the 2nd GermaNet Workshop, 2005*.
- KVH05 Komulainen, V., Valo, A. and Hyvönen, E., A tool for collaborative ontology development for the semantic web. *Proceedings of International Conference on Dublin Core and Metadata Applications (DC 2005)*, Nov 2005.
- LSLH03 Lee, K.-P., Swearingen, K., Li, K. and Hearst, M., Faceted metadata for image search and browsing. *Proceedings of CHI 2003, April 5-10, Fort Lauderdale, USA*. Association for Computing Machinery (ACM), USA, 2003.
- LTS03 Legrand, S., Tyrväinen, P. and Saarikoski, H., Bridging the word disambiguation gap with the help of OWL and semantic web ontologies. *Proceedings of the Workshop on Ontologies and Information Extraction, Europlan 2003*, July 2003, pages 29–35.
- Map95 Maple, A., Faceted access: A review of the literature. Technical Report, Working Group on Faceted Access to Music, Music Library Association, 1995. URL <http://www.musiclibraryassoc.org/BCC/BCC-Historical/BCC95/95WGFAM2.html>

- MB05 Miles, A. and Brickley, D., editors, *SKOS Core Guide*. World Wide Web Consortium, November 2005. URL <http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102/>. W3C Recommendation Working Draft.
- MM00 Moldovan, D. I. and Mihalcea, R., Using WordNet and lexical operators to improve internet searches. *IEEE Internet Computing*, 4,1(2000), pages 34–43.
- MSS⁺01 Maedche, A., Staab, S., Stojanovic, N., Studer, R. and Sure, Y., SEAL — a framework for developing semantic web portals. *Advances in Databases, Proceedings of the 18th British National Conference on Databases*, Jul 2001, pages 1–22.
- MvH04 McGuinness, D. L. and van Harmelen, F., editors, *OWL Web Ontology Language Overview*. World Wide Web Consortium, February 2004. URL <http://www.w3.org/TR/owl-features/>. W3C Recommendation.
- MVL⁺05 Mäkelä, E., Viljanen, K., Lindgren, P., Laukkanen, M. and Hyvönen, E., Semantic yellow page service discovery: The Veturi portal. *Poster paper, 4th International Semantic Web Conference*, Nov 2005.
- NP01 Niles, I. and Pease, A., Towards a standard upper ontology. *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, New York, NY, USA, 2001, ACM Press, pages 2–9.
- NT04 Niles, I. and Terry, A., The MILO: A general-purpose, mid-level ontology. *IKE*, Arabnia, H. R., editor. CSREA Press, 2004, pages 15–19.
- Par04 Parry, D., A fuzzy ontology for medical document retrieval. *CRPIT '04: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, Darlinghurst, Australia, Australia, 2004, Australian Computer Society, Inc., pages 121–126.
- PNL02 Pease, A., Niles, I. and Li, J., The suggested upper merged ontology: A large ontology for the semantic web and its applications. *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, July 2002.
- Pol98 Pollitt, A. S., The key role of classification and indexing in view-based searching. Technical Report, University of Huddersfield, UK, 1998.

- QBHK03 Quan, D., Bakshi, K., Huynh, D. and Karger, D. R., User interfaces for supporting multiple categorization. *INTERACT*, Rauterberg, M., Menozzi, M. and Wesson, J., editors. IOS Press, 2003.
- QHK03 Quan, D., Huynh, D. and Karger, D. R., Haystack: A platform for authoring end user semantic web applications. *Proceedings of the Second International Semantic Web Conference*, 2003, pages 738–753.
- RSC04 Reynolds, D., Shabajee, P. and Cayzer, S., Semantic Information Portals. *Proceedings of the 13th International World Wide Web Conference on Alternate track papers & posters*. ACM Press, May 2004.
- RSdA04 Rocha, C., Schwabe, D. and de Aragão, M. P., A hybrid approach for searching in the semantic web. *Proceedings of the 13th international conference on World Wide Web*, May 2004, pages 374–383.
- Saa04 Saarela, S., Näkymäpohjainen RDF-haku. Master's thesis, University of Helsinki, 2004.
- SDA04 Singh, S., Dey, L. and Abulaish, M., A framework for extending fuzzy description logic to ontology based document processing. *Advances in Web Intelligence, Proceedings of the Second International Atlantic Web Intelligence Conference*, 2004, pages 95–104.
- SH05 Sidoroff, T. and Hyvönen, E., Semantic e-government portals - a case study. *Proceedings of the ISWC-2005 Workshop Semantic Web Case Studies and Best Practices for eBusiness SWCASE05*, Nov 2005.
- SMS02 Sellen, A., Murphy, R. and Shaw, K. L., How Knowledge Workers Use the Web. *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI Letters 4(1)*. ACM, 2002.
- Sta02 Statistics Finland, *Standard Industrial Classification TOL 2002*. Valopaino, Helsinki, 2002. URL http://stat.fi/tk/tt/luokitukset/lk_en/toimiala_02_keh.html. ISBN 952-467-097-6.
- TAAK04 Teevan, J., Alvarado, C., Ackerman, M. S. and Karger, D. R., The perfect search engine is not enough: a study of orienteering behavior in directed search. *Proceedings of the Conference on Human Factors in Computing Systems, CHI*, April 2004, pages 415–422.

- Uni99 United Nations, Statistics Division, *Classification of Individual Consumption by Purpose (COICOP)*. United Nations, New York, USA, 1999. URL <http://unstats.un.org/unsd/cr/registry/regcst.asp?Cl=5>.
- Vil06 Viljanen, K., Monilähteinen suosittelu semanttisessa webissä. Master's thesis, University of Helsinki, March 2006.
- ZM05 Zhang, J. and Marchionini, G., Evaluation and evolution of a browse and search interface: Relation Browser++. *dg.o2005: Proceedings of the 2005 national conference on Digital government research*. Digital Government Research Center, 2005, pages 179–188.
- ZYZ⁺05 Zhang, L., Yu, Y., Zhou, J., Lin, C. and Yang, Y., An enhanced model for searching in semantic portals. *WWW '05: Proceedings of the 14th international conference on World Wide Web*, New York, NY, USA, 2005, ACM Press, pages 453–462.