HGQL: The Hermes Graphical Query Language

Allard Kleijn
303118ak
303118ak@student.eur.nl

Bachelor Thesis
Economics and Informatics Erasmus School of Economics
Erasmus University Rotterdam

Supervisor: Dr. Flavius Frasincar
Co-supervisor: Frederik Hogenboom MSc.

July 13, 2009

Abstract

This paper proposes an extension to the Hermes Framework, allowing the user to create advanced graphical queries. Hermes is a Web-based framework designed to build personalized news services using Semantic Web technologies. It makes use of ontologies for knowledge representation, natural language processing techniques for semantic text analysis and semantic query languages for specifying the desired information. The extension proposed in this paper offers the user the ability to create powerful queries for news items extraction in an intuitive way. The previous method allows only concept selection, however this has been extended with anonymous concepts, conjunctions, disjunctions, negations, text match filters and triple based patterns.

Chapter 1

Introduction

Due to the large amount of news information on the Web and the demand for it ever growing, we are looking for effective ways to search. The news information present on the Web is distributed and possibly heterogeneous. There are many tools available for searching different sources of information, however there always appears to be a trade-off between user interface complexity and language expressiveness. For example SPARQL Protocol and RDF Query Language (SPARQL) [1] is a powerful query language, but not easy to use for a non-expert user. On the easy-to-use side are tools like Magnet [2] and Flamenco [3]. Nevertheless, these tools have expressiveness limitations.

The Hermes Framework is a framework for querying personalized news using Semantic Web technology and Natural Language Processing techniques [4]. HNP retrieves news items from RSS feeds, which are then 'classified', meaning that through word sense disambiguation the meaning of a news word can be identified from the context, possibly representing one ontology concept. These words with semantics are matched against a knowledge base allowing HNP to 'understand' what a news item is about. These 'classified' news items are stored in an OWL[5] ontology , which can be queried by using SPARQL. The user can select concepts from the previously mentioned ontology, which is graphically represented as a graph, in order to form queries.

1.1 Problem and goal

Although Hermes uses the powerful SPARQL 'under the hood', the user is limited to selecting concepts of interest and adding weights to these concepts. Basically this is a disjunctive query; whenever any of the concepts appear in a news item this item is considered relevant. Examples of queries that can not be constructed using Hermes are conjunctive queries, negation or queries with filters. The goal of this paper is to propose an easy-to-use graphical user interface that allows the users to create complex queries for retrieving relevant news from RSS feeds. In addition the following research subquestions can be identified:

- Which existing solutions for graphical query representation are available?
- How well do these existing solutions fit our current goal?
- Which query operators are interesting to be used in the Hermes framework to fit our current goal?
- What aspects are critical in designing an intuitive user interface that fits our current goal?

1.2 Methodology

In order to find answers to the research questions and a solution to fit the goal a methodology is defined. The first step is a literature review to identify key concepts, context and approaches in the field of graphical query languages. The existing methods from the literature are analyzed to identify which concepts and approaches can be used in the Hermes Graphical Query Language(HGQL). Second, a set of features which the HGQL should provide is defined. Third, the components are defined, which form the 'building blocks' for HGQL. Next, the 'grammar' for the HGQL is designed and tested to check whether it allows for the set of features defined in the second step. In addition HGQL shouldn't become too complex.

1.3 Structure

This thesis has 5 chapters which are organized as follows. Chapter 1 provides an introduction and states the problem and goal. Chapter 2 provides an overview of related work in this field. Chapter 3 explains the conceptual framework of Hermes and is divided into four sections, many of these sections describe existing features of the Hermes framework. Section 3.1 discusses classification of news items, Section 3.2 discusses updating of the ontology, Section 3.3 discusses news querying in which HGQL is discussed along with the traditional methods of querying and finally Section 3.4 discusses results presentation. Chapter 4 discusses an implementation of the Hermes framework, more specifically Section 4.1 discusses the traditional querying method, Section 4.2 discusses user profiling, and Section 4.3 discusses the Hermes Graphical Query Language. Chapter 5 presents final conclusions and future work.

Chapter 2

Related work

As mentioned in Section 1.1, Hermes does not support complex graphical queries, however the SPARQL that is the target query language can support complex queries. Graphical interfaces are the most intuitive to use for humans so therefore we look into a way to represent queries in a graphical way. This section is an overview of research on that topic.

2.1 Faceted navigation systems

First are systems that use a construct called facets, such as Flamenco and Magnet that have already been mentioned in Chapter 1. These facets, which can be defined as filters composed of meta data [2, 3]. The user is able to select a facet within a certain domain, which then filters the data set accordingly. On these filtered results the user can stack a second filter. For example within the domain arts the user can select in the facet 'date' the individual facet '17th century' and in the facet 'techniques' the facet 'oil paintings'; the results will be oil paintings from the 17th century. These systems are very limited; it is not possible to do any disjunctions in the queries.

Facets are always used in a 'pipeline', limiting the expressiveness. Faceted navigation user interfaces also pose a problem with schemes for large knowledge bases; once the number of facets becomes higher this technique becomes less intuitive. Although in general facets are intuitive, faceted navigation systems are too limited for implementation in the Hermes Framework.

2.2 Graphical representation of RDF queries

In [6] a technique for building graphical queries for RDF is introduced. This is interesting, because HNP uses RDF and OWL languages as well. RDF is built up from triples in the following format: subject predicate object. The representation discussed in [6] uses rounded rectangles with a predicate and optionally object in it to visualize queries, as can be seen in Fig. 2.1. It also allows multiple rounded rectangles to be linked returning only results that match both queries. It also supports object nesting making it possible for different resources to be linked to each other.

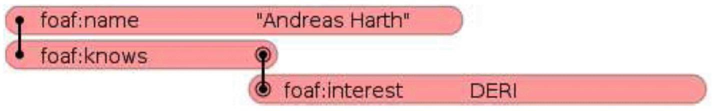


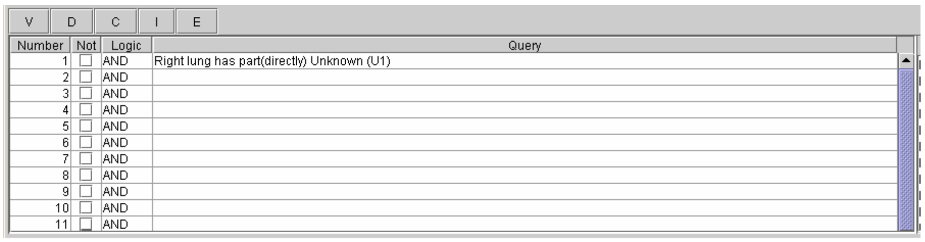
Figure 2.1: Example of RDF graphical representation discussed in [6]

This language has the following structure: every ‘block’ is used to represent a triple possibly containing subject, predicate and object. Blocks can be connected at the same level, as in “foaf:name “Andreas Harth” and “foaf:knows””, which creates a conjunctive query of these two blocks. Blocks can also be chained by connecting one block at the object to another block at the subject, in Fig. 2.1 that is “foaf:interest DERI”, this creates a chained query. A double circle denotes what should be retrieved. As can be seen in Fig. 2.1 this specific case of this method of graphical representation has the following meaning: Get everyone that is interested in DERI and known by Andreas Harth. This technique is interesting in the context of the extension to the Hermes Framework, because the language’s expressiveness. It allows chaining, filters and it’s built for RDF, which is also the case for the Hermes Framework. RDF’s triple pattern is also intuitive to humans as the subject-predicate-object represents the structure of a basic English sentence (subject-verb-predicate) [7]. For example the query “Subject hasName ‘Andreas Harth’” is very similar in structure to the basic English sentence “Man buys dog”; making the queries easy to read for humans. However there is no functionality to create disjunctive queries (‘OR-queries’)[6].

2.3 Emily

Emily [7] takes a different approach in graphically representing queries. This tool was created as a graphical query engine for Protégé. Emily displays three lists of items; the first list contains subjects, the second contains relations, and the third contains objects. Both subject and object lists contain classes from the knowledge bases represented in a hierarchical tree. The user specifies all three fields (although subject and/or object can be set to ’Unknown’). Once the user makes a query it is stored in another list. The user can then perform some logic on the queries (and/or and not) making it possible to combine queries with certain logical operators, which can be seen in Fig. 2.2.

Emily’s interface provides a simple way to create queries (using the subject-relation-object browser). Emily is probably less suitable for very large knowledge bases since the lists of subjects, objects and relations would become very large, however a search function is implemented giving Emily more potential for large knowledge bases. Emily allows many complicated queries to be created,


V D C I E

Number	Not	Logic	Query
1	☐	AND	Right lung has part(directly) Unknown (U1)
2	☐	AND	
3	☐	AND	
4	☐	AND	
5	☐	AND	
6	☐	AND	
7	☐	AND	
8	☐	AND	
9	☐	AND	
10	☐	AND	
11	☐	AND	

Figure 2.2: Emily's logic panel

however the part of the interface that allows the user to add logic to queries appears to be unintuitive as can be seen in Fig. 2.2. It is not clear how the logical and/or (in the column 'Logic') operate between queries or elements of these queries. In addition the user is provided with a set of buttons "V, D, C, I, E", which have no clear meaning either. Both of these unclarities are not discussed in [7].

2.4 RDF-GL

RDF-GL [8] is a SPARQL-based graphical query language for RDF. RDF-GL's component consists of rounded rectangles referred to as boxes, circles, arrows and different colors. Orange boxes contain information about the query; sort results in ascending order for example. Purple boxes represent subjects or objects. Green boxes denote filters for subjects/objects. Purple circles are used to state that something is optional and blue circles are used to display unions. Black arrows stand for relations and gray arrows for optional statements. Yellow and red arrows are used to point to relations belonging to a union block; yellow arrows pointing to the first block and red to the second.

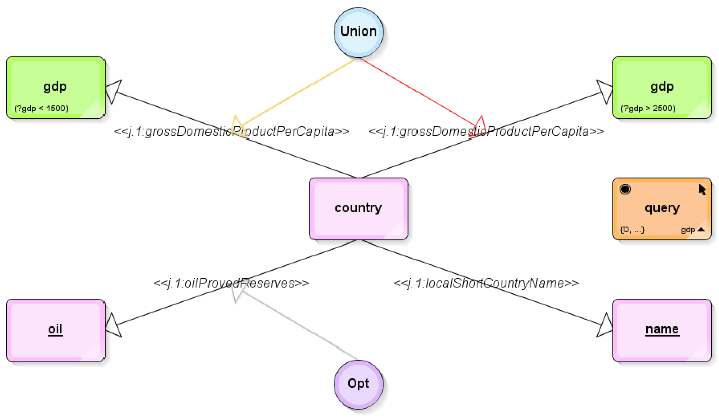

Union
gdp
(?gdp < 1500)
gdp
(?gdp > 2500)
<<j.1:grossDomesticProductPerCapita>>
<<j.1:grossDomesticProductPerCapita>>
country
query
{0, ...}
gdp
<<j.1:oilProvedReserves>>
<<j.1:localShortCountryName>>
oil
name
Opt

Figure 2.3: An example of RDF-GL

RDF-GL is very powerful; allowing the user to create a large set of SPARQL queries, however RDF-GL is unintuitive as one can see in Fig. 2.3. There are many different colors and shapes which makes the overview of the query difficult.

2.5 GLOO

Another query visualization technique is GLOO [9], in which concepts, individuals and the logical operators AND/OR can be specified. These elements are visualized by ovals or squares. This allows the user to design queries in a fairly intuitive way.

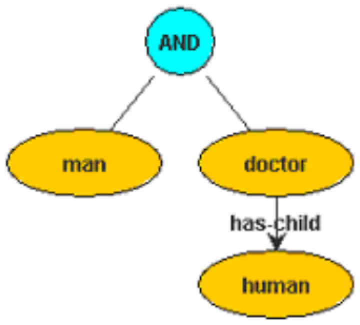


Figure 2.4: An example of GLOO

We can see that this query attempts to retrieve a result which is both a man and a doctor and this person has a child. This representation is easy to understand for non-expert users.

GLOO is a powerful graphical query language, allowing the user to make complex queries in an intuitive way. However in [9] there is no method discussed that allows the implementation of filters as in RDF-GL.

Chapter 3

Hermes Framework

Hermes [4] is a framework aimed at personalizing news items. The Hermes framework makes use of RSS feeds as input. The news items coming from these feeds are processed and classified using a knowledge base. The user can choose concepts from the knowledge base to query the input news items. Based on these queries, the system returns the news items of interest to the user.

As an example, the framework makes use of a knowledge base about NASDAQ companies. This ontology consists of concepts like companies, products, competitors, CEO's, etc. Once the news items are classified, they are stored in a news ontology together with their meta data.

The Hermes framework features an architecture of various phases. First, the Classification phase classifies the news items retrieved from the feeds. The classified news items are stored in the news ontology. The system can search the news for events affecting the knowledge base and possibly change the knowledge base according to these events if the user approves these updates. This happens in the Update Ontology phase. The News Querying phase allows the user the opportunity to search the news, either by using selecting concepts direclty or indirectly; based on his profile. Finally, the Results Presentation phase sorts the news in a logical order based on the relevance the items have to the user's query. We will discuss these different phases in the next sections. We focus mainly on the extensions proposed in this thesis and briefly present previous work for completion reasons.

3.1 Classification

In order to be able to query the news, the news items should first be classified. This is done using a series of Natural Language Processing (NLP) steps. The classification process searches the news items for concepts from the knowledge base. The news items are stored in an ontology together with the concepts found and how often these concepts occurred in the news item. This allows for faster querying as opposed to classifying the news items 'on the fly'.

3.2 Updating Ontology

After the Classification phase we continue with the phase where we update the knowledge base. In this step, the system searches the news items for specific events which affect the knowledge base. If any event of the required type occurred, the system will present this to the user. The user has to judge if the proposed changes in the knowledge base are correct, and if so, let the system update the ontology.

3.3 News Querying

In the news querying phase the user should state his interests. We distinguish three different methods of news querying, first we discuss the traditional method, and second we discuss the user profile-based method and finally HGQL.

3.3.1 Traditional

In the traditional way, which was part of the original Hermes framework, the user can state his interests by selecting the concepts he is interested in from the knowledge base. The system visualizes the knowledge base to the user as a graph, so the user can observe all available concepts he can select from. The user can then select all the concepts of interest and add them to the search graph. While adding these concepts, he can also decide to add all concepts with a specific relation to the concepts of his choice. This is an easy way of searching, as the user doesn't need to know all concepts related to his interests because all the relevant relations are stored in the knowledge base.

The user also has the ability to specify a time constraint. The time constraints allow the user to specify a specific interval in which the news items have to be published in order for them to be returned.

When the user has stated his interests and the time constraint, the system will search the news. The news is queried using a query engine which supports the chosen query language. The query will only return the news items which contain one or more of the concepts of interest.

3.3.2 Profile-based

In the profile-based approach the user has to log in on his profile to be able to store his interests. He still has to select the concepts of interest from the knowledge base graph. The concepts of his choice are stored in a graph, called the user profile. In addition, the user has to add importance weights to the chosen concepts. We chose to define such a weight to be any integer value between 1 and 5. The value 1 means "Slightly important" in this approach, while the value 5 means "Very important". The weights assigned to the concepts are used in the Results Presentation phase to compute the news item's relevance to the user's query.

3.3.3 Hermes Graphical Query Language

The main focus of this thesis is to provide a new method of querying news items than before; allowing the user to construct more complex queries, while ensuring the language remains easy-to-use. The query language is called Hermes Graphical Query Language (HGQL). A query in the HGQL is represented as a directional graph. The elements and structure are described in the following sections.

HGQL elements

Elements in HGQL can be divided into two categories: nodes and edges. Nodes can represent concepts, wild cards, operators, and relations and take the form of rounded rectangles in different colors. Edges represent connections between nodes and take the form of lines with arrows at one end.

Concepts The concepts used in HGQL are the same concepts as described in Sect. 3.3.1. In order to keep the Hermes Framework coherent the concept node's colors are kept the same (purple) as previous querying methods. Concepts are denoted by noun phrases.

Wild cards The wild cards are a construct in the HGQL that stand for a group of nodes, which represents either concepts or relations (partly) unknown to the user. The wild card can be either 'unknown', which has the same meaning as in Emily [7] or a text match. For example the HGQL query "Google Unknown_Relation Yahoo" will return any news items in which a relation between Google and Yahoo exists. The text match provides the user with the ability to find concepts or relations for which he doesn't know the exact name. For example the user is interested in Apple's CEO Steve Jobbs, but he can only remember 'Jobbs'. The user can use a text match node for "Jobbs" and the concept with the closest match will be used in the query. From this point when referring to concepts this will include wild cards of type concept and when referring to relations, wild cards of type relation. Instances of the wild card category are represented in the color of the type of node they match, which is purple for concepts and green for relations.

Operators The operators are a group of nodes in HGQL that allows the user to add logic to queries. There are two types of operators: intra-query and inter-query. Intra-query operators support logic within queries and can be either: conjunction (and), disjunction(or), and negation (not). The reason for using these specific operators is that the combination of them can create any possible first order logic formula. Instances of the intra-query operators category are denoted as light blue nodes. Inter-query operators support logic (conjunction and disjunction) between different queries, allowing the user to process multiple queries simultaneously. Inter-query operators connect to the top node (the node that has no incoming edges) of the query; if the query is a (chained) triple-based pattern query the inter-query operator connects to the first predicate or root of predicates. Instances of inter-query operators are denoted as yellow nodes.

Relations The relations allow the user to employ HGQL's triple based pattern querying. By creating a query with the structure *concept* $\rightarrow$ *relation* $\rightarrow$ *concept*, news items matching that pattern will be returned. The possible instances of a relation are extracted from the knowledge base. Relations are represented by verb phrases. Instances of the relation category are denoted as green nodes.

Edges The edges provide connectivity between nodes and consist of two types: logical connectivity edges and pattern connectivity edges. Logical connectivity edges allow for the connecting of operators to concepts and relations. The arrows for logical connectivity edges always point from the operator node to concepts, wild cards, or relations and are colored light gray.

Pattern connectivity edges allow for the creation of the pattern structure as can be seen in Fig. 3.2. This type of edge can connect the concepts, wild cards, and relations to form the *Subject* $\rightarrow$ *Predicate* $\rightarrow$ *Object* pattern. If multiple subjects exist logical connectivity edges connect these subjects to an operator, which can be connected to a relation node or a node that is the root to multiple relations (Fig. 3.2 is a good example). The direction of a pattern connectivity edges determines the pattern, the relation(s) need(s) an incoming edge which connects to the subject(s) and an outgoing edge which connects to the object(s). Pattern connectivity edges are colored black.

HGQL structure

As mentioned in Sect. 3.3.3 a query in HGQL is represented as a directional graph. Three types of queries can be identified, for each one the structure will be discussed.

Concepts-only queries The concept-only queries are the most simple of types; they consist of one or more concept nodes connected by operators (for more than one node). This approach comes close to the previous approaches of Hermes, however the previous approaches only allowed for disjunctive combination of concepts. This query type has the following structure: an and/or operator as the root node and all concepts connected to the root node. The direction of the edges is towards the non-root nodes. It is allowed to perform logic on concepts before connecting them to the root node. An example of a complex concepts-only query can be seen in Fig. 3.1. The meaning of this query is: Retrieve all news items in which *Dell* is mentioned and either *Cisco* or *Mac OS* and *AMD* is not mentioned.

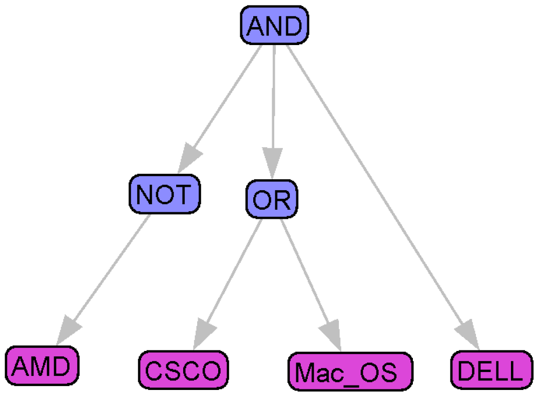


Figure 3.1: An example of a concepts-only query

Triple based pattern queries The triple-based pattern queries allow the user to query for patterns within news items. A pattern has the following structure: *Subject* → *Predicate* → *Object*. This pattern, also employed in RDF, is used, because: "...it mimics the structure of a simple English sentence (subject-verb-predicate)." [7]. In order to maintain HGQL's usability this structure is used for triple based pattern queries. A simple query of this type would be *concept* → *relation* → *concept*, however it is possible to use logical operators in any of these three fields. In order to maintain the *Subject* → *Predicate* → *Object* structure a triple based pattern query always has pattern connectivity edges pointing from the subject(s) towards the relation(s) and from there towards the object(s). When a triple based pattern query has multiple subjects, relations, and objects it can be broken down into three trees, which are connected to one another at the root node by pattern connectivity edges, where the root of the subjects connects to the root of the relations, which in turn connects to the root of the objects. An example can be seen in Fig. 3.2.

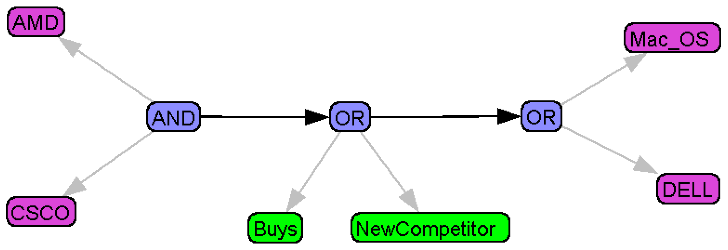


Figure 3.2: An example of a triple based pattern query

This query has the following meaning: Retrieve all news items which match the patterns $AMD \rightarrow NewCompetitor/Buys \rightarrow Dell/MacOS$ and $CISCO \rightarrow$

NewCompetitor/Buys → *Dell/MacOS*. A news item needs to contain both of these patterns in order to be returned.

Chained The chained queries are a specific subset of triple based pattern queries; they have all the same properties except it is possible to use the object of one triple pattern as the subject of another triple pattern in a similar way as in [6].

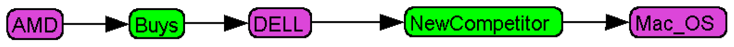


Figure 3.3: An example of a chained query

As can be seen in Fig. 3.3 a chained query is basically two triples fused together. This specific query would have the following meaning: Retrieve all news items that match the pattern *AMD* → *NewCompetitor* → *Dell* and the pattern *Dell* → *Buys* → *MacOS*.

3.3.4 HGQL Grammar

HGQL's grammar is a set of rules on which a query can be validated. Some of the grammar rules are general some are specific for a query type as discussed in the previous section. If any of these rules can not be validated the query is invalid. In the following sections these rules are discussed. The presentation order of the rules is the order in which they should be executed, because of the rules dependencies. For example, for a query that holds no subjects there is no point in checking the other nodes for correct connection conditions.

Floating nodes

In HGQL it is not allowed for a node to float (having no edges connecting it). Any query that holds a floating node is by definition invalid. When the user creates multiple queries they have to be interconnected; it is not allowed to have multiple subgraphs as a query.

Subjects or objects

For every query there should be at least one concept node. In addition if a query has a relation node the query is considered a triple based pattern query and therefore should have at least one subject and one object. Subjects and objects can be identified by the direction of the pattern connectivity edges, as they point towards the object.

Connections

HGQL allows nodes to be connected to almost all other type of nodes, however, there are a few exceptions. First, nodes are not allowed to connect to a node of the same type, with the exception of nodes of type operator. Second, when a concept node is considered a subject it is not allowed to connect that node to a node which is considered an object.

Connection conditions

For every node the number outgoing and incoming edges can be verified. The following conditions hold for all but chained queries. The actual concepts are considered (not the root) when referring to subject, object or sub/object in the following tables. The same principle goes for relations. The node type AND/OR Operator in the following tables are both intra- and inter-query operators, because they share the same conditions.

Node type	Incoming logical edges	Outgoing logical edges
AND/OR Operators	0-*	1-*
NOT Operator	0-*	1
Subject	0-1	0
Object	0-1	0
Relations	0-1	0

Table 3.1: Logical connectivity edge conditions for non-chained queries

Node type	Incoming pattern edges	Outgoing pattern edges
AND/OR Operator	0-1	0-1
NOT Operator	0-1	0-1
Subject	0	0-1
Object	0-1	0
Relations	0-1	0-1

Table 3.2: Pattern connectivity edge conditions for non-chained queries

A value of 1-* means at least 1, but can be any integer value larger than 0. A value of 0-1 is either 0 or 1.

When a node of type concept is found which has both incoming and outgoing pattern connectivity edges the query is automatically assumed to be a chained query. This type of node is from now on referred to as sub/object, because it is both a subject and object. A chained query has the following set of conditions.

Node type	Incoming logical edges	Outgoing logical edges
AND/OR Operator	0-*	1-*
NOT Operator	0-*	1
Subject	1-*	0
Object	1-*	0
Sub/object	0-*	0
Relations	0-1	0

Table 3.3: Logical connectivity edge conditions for chained queries

Node type	Incoming pattern edges	Outgoing pattern edges
AND/OR Operator	0-*	0-*
NOT Operator	0-1	0-1
Subject	0	0-*
Object	0-*	0
Sub/object	0-*	0-*
Relations	0-1	0-1

Table 3.4: Pattern connectivity edge conditions for chained queries

3.4 Results Presentation

3.4.1 Traditional methods

The most important part of the results presentation is the relevance sorting. Relevance sorting is an important feature of news personalization. If this feature is not implemented in the system, the user would struggle to filter the news items which are most important for him. The relevance sorting moves the most relevant news items to the top of the result list. For every retrieved news items a relevance score is calculated [4]. This score is normalized to provide a uniform range (0 to 100%) for the relevance. The results are then sorted in a descending order.

3.4.2 HGQL

Once the querying of news items is complete the results can be processed. For the current HGQL framework this means an unsorted list of results is presented to the user.

Chapter 4

Hermes News Portal

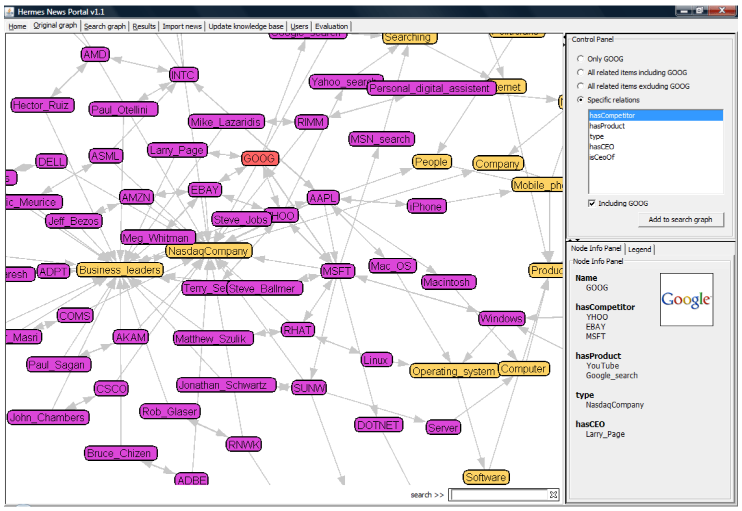
In this section we discuss the implementation of the extensions that we proposed to the Hermes framework. The Hermes News Portal (HNP) is an implementation of the Hermes framework. This implementation allows the user to employ all the features of the Hermes framework. We discuss the parts of the HNP we added or changed in the following sections. Please note that the HNP is just one possible implementation of the Hermes framework.

As ontology language, we have chosen OWL [5] because this language is supported by the W3C and it offers some useful additional features we need in HNP. Some examples of these features are the possibilities to describe disjoint classes, cardinality, and symmetry. We use OWL ontologies to store the classified news items, the knowledge base, the update rules, and the user profile. The query language we chose to query OWL ontologies is SPARQL [1]. SPARQL is also supported by W3C as the query language for RDF languages including OWL. To deal with the time constraints we used an extension of SPARQL with temporal primitives, tSPARQL [4]. To represent the ontology in a graph we use Prefuse [10].

The Java programming language is chosen for the implementation of the whole program. This is because there are many libraries for manipulating, reasoning with, querying and visualizing ontologies available. The libraries which are used, are GATE [11] for the Natural Language Processing steps in the classification phase, ARQ [12] to execute the SPARQL queries, SPARQL/update [13] to update ontologies, and OWL2Prefuse[14] for the visualization of the knowledge base.

4.1 Traditional querying

The implementation of the traditional method of querying provides the user with two graphs. The first graph, called “Original graph”, represents all the concepts currently stored in the knowledge base. The user can select concepts directly by clicking them, or indirectly by selecting concepts related to another concept in the top-right panel, as seen in Fig. 4.1.



Control Panel

( ) Only GOOG
( ) All related items including GOOG
( ) All related items excluding GOOG
(•) Specific relations

hasCompetitor
hasProduct
type
hasCEO
isCeoOf

☑ Including GOOG

Node Info Panel

Name
GOOG

hasCompetitor
YHOO
EBAY
MSFT

hasProduct
YouTube
Google_search

type
NasdaqCompany

hasCEO
Larry_Page

Figure 4.1: The original graph visualizing a part of the knowledge base

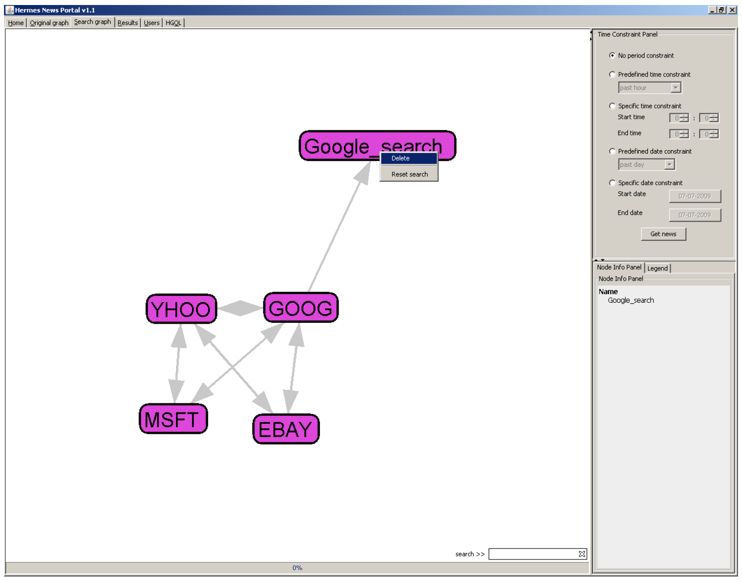
When the user presses the 'Add to search graph' button the selected concepts are placed in the "Search graph". This graph offers the user the ability to check his concepts to query for, add time constraints, and remove concepts, as seen in Fig. 4.2. When the user is satisfied with the selected concepts he can press the 'Get news' button, which instructs HNP to query for results. Any news item that contains any of the query concepts is returned.

A relevance score is calculated [4] and normalized to sort the results in order to have the most relevant items on top of the results list, as can be seen in Fig. 4.3.

4.2 User Profiling

The implementation of the user profiling service provides the user with the ability to create an account. When he is logged in on this account, for every search he does using the traditional querying method, the unique concepts are stored in his profile. This profile is represented as a graph. By right-clicking on the 'profile concepts' the user can change the importance of them to a value of 1 to 5 (1 being the least important). The importance of a concept is represented by the size of the node, giving the user a quick overview.

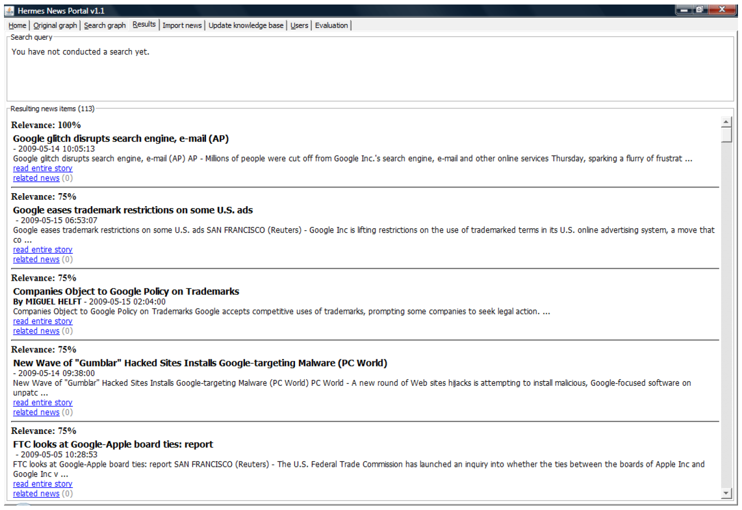
By pressing the 'Get news based on profile' button the HNP queries the news for for the concepts contained by the profile. In the calculation of the relevance to the query the concept weights (importance) are taken into account, giving



[✓ No period constraint]
Predefined time constraint: past hour
Specific time constraint
Start time 0 : 0
End time 0 : 0
Predefined date constraint: past day
Specific date constraint
Start date 07-07-2009
End date 07-07-2009

Node Info Panel
Name
Google_search

Figure 4.2: A search graph showing 5 concepts and the right-click menu


Hermes News Portal v1.1
Home | Original graph | Search graph | Results | Import news | Update knowledge base | Users | Evaluation

Search query
You have not conducted a search yet.

Resulting news items (113)

Relevance: 100%
Google glitch disrupts search engine, e-mail (AP)
- 2009-05-14 10:05:13
Google glitch disrupts search engine, e-mail (AP) AP - Millions of people were cut off from Google Inc.'s search engine, e-mail and other online services Thursday, sparking a flurry of frustrat ...
read entire story
related news (0)

Relevance: 75%
Google eases trademark restrictions on some U.S. ads
- 2009-05-15 06:53:07
Google eases trademark restrictions on some U.S. ads SAN FRANCISCO (Reuters) - Google Inc is lifting restrictions on the use of trademarked terms in its U.S. online advertising system, a move that co ...
read entire story
related news (0)

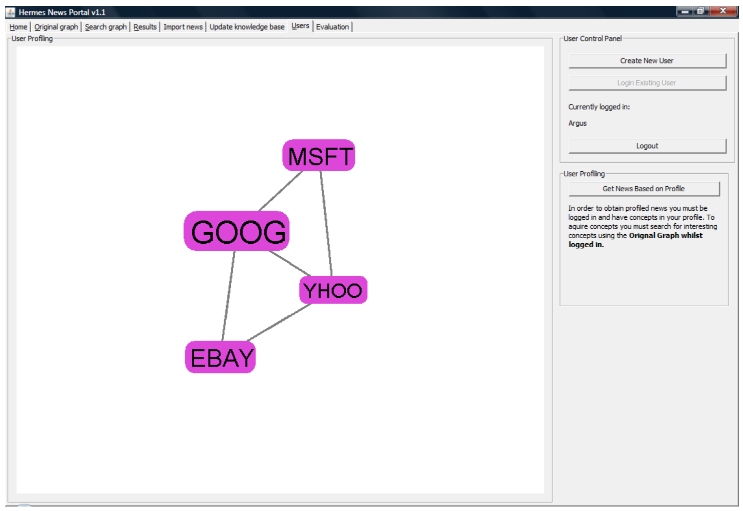
Relevance: 75%
Companies Object to Google Policy on Trademarks
By MIGUEL HELFT - 2009-05-15 02:04:00
Companies Object to Google Policy on Trademarks Google accepts competitive uses of trademarks, prompting some companies to seek legal action. ...
read entire story
related news (0)

Relevance: 75%
New Wave of "Gumblar" Hacked Sites Installs Google-targeting Malware (PC World)
- 2009-05-14 09:38:00
New Wave of "Gumblar" Hacked Sites Installs Google-targeting Malware (PC World) PC World - A new round of Web sites hijacks is attempting to install malicious, Google-focused software on unpatc ...
read entire story
related news (0)

Relevance: 75%
FTC looks at Google-Apple board ties: report
- 2009-05-05 10:28:53
FTC looks at Google-Apple board ties: report SAN FRANCISCO (Reuters) - The U.S. Federal Trade Commission has launched an inquiry into whether the ties between the boards of Apple Inc and Google Inc v ...
read entire story
related news (0)

Figure 4.3: The results tab

the user more flexibility than the traditional querying method. An example of the user profiling interface can be seen in Fig. 4.4.



In order to obtain profiled news you must be logged in and have concepts in your profile. To aquire concepts you must search for interesting concepts using the **Orignal Graph whilst logged in.**

Figure 4.4: An example of a user profile with 4 concepts with different weights

4.3 HGQL

This section provides a view of an implementation of HGQL in HNP. This implementation specifically uses the packages Prefuse [10], for the visualization of the data and ARQ for the execution of SPARQL queries. This section discusses the implementation of HGQL in HNP, more specifically the user interface, query validation, and querying.

4.3.1 User interface

The user interface for HGQL can be broken down into two parts: the query graph, which visualizes the query using Prefuse and a panel that offers the user a set of controls. Fig. 4.5 shows the interface with an example query graph. This example query consist of two queries linked by an inter-query operator.

As can be seen in Fig. 4.5 the 'Controls' panel holds three types of controls: node creation, node manipulation, and query execution.

Node creation The sub-panels 'Query operators', 'Relations', and 'Concepts' allow for the creation of the types of nodes discussed in Section 3.3.3. The 'Query operators' panel is straightforward; pressing a button will create a node of that type. By default an operator is an intra-query operator, however the user can right-click an operator node and change its type to inter-query operator. The 'Relations' panel offers the user a drop down list of all relations that exist within the knowledge base. In addition the user is provided with two buttons that allow for the creation of relation wild cards, which can either be unknown or text match. For text match the user has to fill in the text he wants to

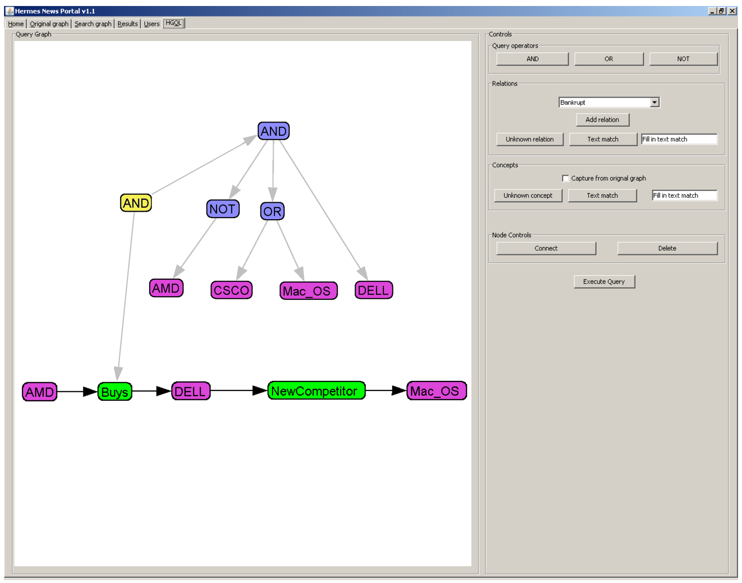


Figure 4.5: The HGQL user interface showing two queries connected with an inter-query operator

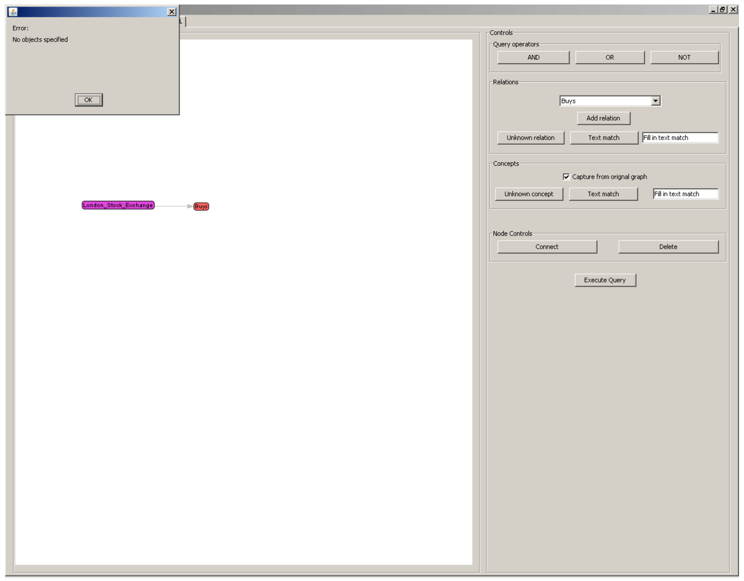
match in the text field and afterwards press the 'Text match' button. The 'Concepts' panel offers the same wild card functionality as the relations panel, but for concepts. No button for direct creation of concept nodes is available, since concepts are to be selected from the original graph (a drop-down list of concepts would be too large to be easily manipulated by the user. After the user checks the box "Capture from original graph" all concepts that the user places in the search graph are also placed in the HGQL graph.

Node manipulation The panel 'Node controls' allows the user to delete and connect nodes. The user can select a set of nodes (holding the Control-key allows the user to select multiple nodes) and press 'Delete'; this will delete all selected nodes and all edges that are connected to these nodes. Connecting nodes has a few more restrictions than deleting; the user is only allowed to connect (select) two nodes and the order of the selection determines the direction of the edge. First selecting node A and then selecting node B followed by pressing the 'Connect' button will create the pattern $A \rightarrow B$. By default HNP creates a logical connectivity edge, however the user can right-click any edge and change it to either a logical connectivity or a pattern connectivity edge.

Query execution Pressing the button 'Execute query' will cause HNP to start validating the query, display an error message when the query is invalid or attempt to execute the query if possible, after which the HNP will switch to the 'Results' tab where the user can view the results.

4.3.2 Query validation

The query validation for HGQL in HNP processes all the rules described in Sect. 3.3.4, if any of these rules can not be validated HNP will not execute the query. In addition, it is allowed for the user to create multiple queries in one graph, however each of these queries have to be connected by inter-query operators and each individual query has to be valid in order for the HNP to execute them. Error messages are presented to the user via a pop-up window. Fig. 4.6 shows what happens when the user creates an invalid query.



Error:
No objects specified

Figure 4.6: An error message in HNP

4.3.3 Querying

This section describes the process of executing a query using HGQL in HNP.

Sub-queries As mentioned in Sect. 4.3.2 it is allowed to create multiple queries in the HGQL query graph. HNP splits the graph to produce a list of sub-queries, which can be processed individually. For every sub-query a SPARQL query is produced, executed and depending on the inter-query operators used added to the list of results, which is presented to the user after all sub-queries have been processed.

HGQL to SPARQL translation In order to describe HGQL's translation to SPARQL a few cases are discussed. Consider the HGQL query shown in Fig.

4.7.

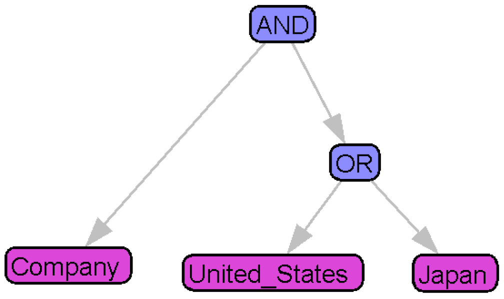


Figure 4.7: A simple query in HGQL

When this HGQL query is translated to SPARQL the result is as follows:

```
PREFIX news: <http://www.hermes.com/news.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX kb: <http://www.hermes.com/knowledgebase.owl#>

SELECT DISTINCT ?news
WHERE
{
?news rdf:type news:News.
?news news:relation ?relation1.
?news news:relation ?relation2.
?relation1 news:relatedTo ?concept1.
?relation2 news:relatedTo ?concept2.
FILTER(?concept1 = <http://.../knowledgebase.owl#Company>
&& (?concept2 = <http://.../knowledgebase.owl#Japan>
|| ?concept2 = <http://.../knowledgebase.owl#United_States>)
)
```

The 'PREFIX' commands allow for the use of name spaces. 'SELECT DISTINCT ?news' means that only unique news items will be selected as a result. The 'WHERE' clause specifies the conditions for the news items to be selected. The most important part of this translation is the 'FILTER' statement; this allows the query to filter the results for certain conditions. In this case we want one concept to be *Company* and the other either *Japan* or the *United States*. Two examples of returned news items from our news ontology are:

Panasonic slumps to $4 billion yearly loss (AP) AP - Panasonic Corp. slumped deep into the red last fiscal year, joining the expanding club of big Japanese brands shellshocked by their rapid descent from cash cow to money loser.

Advertising: A Tech Companys Campaign to Burnish Its Brand Intels new campaign beginning Monday in the U.S. is the companys first to focus on its brand rather than its products, and celebrates the companys role in the future.

This query returns all news items in which a company is mentioned and either Japan or the United States. In the first example, Panasonic is considered a company and ‘Japanese’ links to the concept Japan. In the second example, ‘Company’ is found directly in the text and U.S. is considered to be the United States.

For the second case we look at a triple based pattern query (Fig. 4.8). For this type of query a new class was made in the news ontology. This class is called ‘Pattern’ and holds the subject, object, and predicate of a pattern.

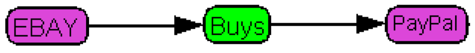


Figure 4.8: A triple based pattern query in HGQL

When this HGQL query is translated to SPARQL the result is as follows:

```
PREFIX news: <http://www.hermes.com/news.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX kb: <http://www.hermes.com/knowledgebase.owl#>

SELECT DISTINCT ?news
WHERE
{
?news rdf:type news:News.
?news news:relation ?relation.
?relation news:relatedTo ?pattern.
?pattern news:subject ?subject.
?patern news:predicate ?predicate.
?pattern news:object ?object.
FILTER(?subject = <http://.../knowledgebase.owl#EBAY>
&& ?predicate=<http://.../knowledgebase.owl#Buys>
&& ?object =<http://.../knowledgebase.owl#PayPal>)
}
```

Again distinct news items are returned, but this time only relations that relate to a pattern are selected. For these items the subject, predicate, and object are selected and compared against the subject, predicate and object from the query. As can be seen this filter has conjunctive semantics (&&), because the pattern has to match on all three fields. An example of a returned news item is:

EBay to Buy PayPal, a Rival in Online Payments The online auction giant eBay announced plans today to acquire PayPal, a rapidly growing start-up that lets people make payments via e-mail. The move will let eBay take a bigger share of many sales made on its own site, and to expand its business to earn money on tens of thousands of transactions made elsewhere on the Internet.

Chapter 5

Conclusions and future work

5.1 Conclusions

HGQL provides a graphical language to represent queries within the context of news personalization provided by Hermes. A set of building blocks; concepts, relations, operators, wild cards, and edges allow the user to construct queries without any knowledge of SPARQL. HGQL provides the possibility to create queries consisting of only concepts and logical operators, triple-based pattern queries and chaining of queries.

The implementation of HGQL in HNP provides the user with a tab, allowing him to create, connect, and delete nodes. Concept nodes can be selected in HNP's 'Original graph'; operators, relations, wild cards, and edges can be created from the HGQL interface. The query is displayed as a graph with colored nodes, while a panel on the right side provides the user with buttons for node creation, manipulation and the execution of the query. Buttons are grouped to decrease interface complexity. When executing a query it is first automatically validated according to the rules specified in the HGQL grammar. Invalid queries produce a pop-up screen, instructing the user which rule couldn't be validated.

5.2 Future work

The current framework for the HGQL provides no sorting of the results after they have been retrieved. A sorting algorithm would be a major improvement to the HGQL. Whereas the previous sorting algorithm only copes with disjunctive semantics, an algorithm for HGQL should be able to cope with conjunction, disjunction, and negation.

Another improvement to HGQL would be the possibility to add weights to concepts in HGQL queries (similar to user profiling). This allows the queries to become more flexible, especially for large queries that hold many concepts.

Finally, in order to prove HGQL's ease of use, a usability test should be performed. By letting users make HGQL queries in HNP, data about their mistakes and time taken per query can be gathered, in order to prove HGQL's usability.

Bibliography

[1] Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C candidate recommendation 14 june. from http://www.w3.org/TR/rdf-sparql-query/ (2007)

[2] Sinha, V., Karger, D.R.: Magnet: supporting navigation in semistructured data environments. In: SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, ACM Press (2005) 97–106

[3] K.P. Yee, K. Swearingen, K.L., Hearst, M.: Faceted metadata for image search and browsing. In: CHI '03: Proceedings of the conference on Human factors in computing systems, ACM Press (2003) 401–408

[4] Frasincar, F., Borsje, J., Levering, L.: A semantic web-based approach for building personalized news services. In: International Journal of E-Business Research **5**(3) (2009) 35–53

[5] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Stein, P.F.: OWL web ontology language reference. W3C recommendation. from http://www.w3.org/TR/owl-ref (2004)

[6] Harth, A., Kruk, S.R., Decker, S.: Graphical representation of rdf queries. In: WWW, ACM (2006) 859–860

[7] L.D. Detwiler, C. Rosse, L.S.: An intuitive graphical query language for protégé knowledge bases. from http://protege.stanford.edu/conference/2004/abstracts/Detwiler.pdf (2004)

[8] F. Hogenboom, V. Milea, F.F., Kaymak, U.: RDF-GL: A SPARQL-based graphical query language for RDF. In: Emergent Web Intelligence. (2009 (to appear))

[9] Fadhil, A., Haarslev, V.: Gloo: A graphical query language for owl ontologies. In: OWLED. Volume 216., CEUR-WS.org (2006)

[10] Heer, J.: Prefuse, an information visualisation toolkit. from http://prefuse.org (2009)

[11] Cunningham, H.: Gate, a general architecture for text engineering. Journal Computers and the Humanities **36**(2) (2002) 223–254

[12] Jena Development Team: ARQ, a SPARQL processor for jena. from http://jena.sourceforge.net/ARQ (2009)

[13] Seaborne, A., Manjunath, G.: SPARQL/update - a language for updating RDF graphs. from http://jena.hpl.hp.com/afs/SPARQL-update.html (2009)

[14] Borsje, J.: Owl2Prefuse. from http://owl2prefuse.sourceforge.net (2009)