2007

A Hybrid Approach to Retrieving Web Documents and Semantic Web Data

Trivikram Immaneni
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/220

This Dissertation is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu, library-corescholar@wright.edu.
A HYBRID APPROACH TO RETRIEVING WEB DOCUMENTS AND
SEMANTIC WEB DATA

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

By

TRIVIKRAM IMMANENI
M.S., Wright State University, 2001

2007
Wright State University

______________________________
Krishnaprasad Thirunarayan, Ph.D.
Dissertation Director

______________________________
Thomas Sudkamp, Ph.D.
Director, Computer Science and Engineering Ph.D. Program

______________________________
Joseph F. Thomas, Jr., Ph.D
Dean, School of Graduate Studies

Committee on
Final Examination

______________________________
Krishnaprasad Thirunarayan, Ph.D.

______________________________
Nikoloas Bourbakis, Ph.D.

______________________________
Mateen M. Rizki, Ph.D.

______________________________
Arijit Sengupta, Ph.D.

______________________________
Raj Bhatnagar, Ph.D.
ABSTRACT

Immaneni, Trivikram, Department of Computer Science and Engineering, WSU, 2007
Title: A Hybrid Approach to Retrieving Web Documents and Semantic Web Data

The Semantic Web has been evolving into a property-linked web of RDF data, conceptually divorced from (but physically housed in) the World Wide Web of hyperlinked documents. Data Retrieval techniques are typically used to retrieve data from the Semantic Web while Information Retrieval techniques are used to retrieve documents from the Hypertext Web. Conceptually unifying the two webs enables the exploitation of their interconnections resulting in benefits to both data and document retrieval. Towards this end, we present the Unified Web model that integrates the two webs and formalizes the structure and the semantics of their interconnections. We present a hybrid approach to retrieving data and documents that is enabled by this unification. We specify the Hybrid Query Language that embodies the approach and the system SITAR that implements it.
# Table of Contents

1 Introduction .............................................................................................................................................1

1.1 Contributions ....................................................................................................................................2

1.2 Outline ...............................................................................................................................................3

1.3 Conventions .........................................................................................................................................3

1.3.1 Hypertext Web and the Semantic Web ..................................................................................4

1.3.2 URI vs. URL ..............................................................................................................................4

1.3.3 Formal Query vs. Keyword Query .........................................................................................5

2 The Semantic Web ..................................................................................................................................6

3 Related Research .....................................................................................................................................13

3.1 RDF Storage Systems ......................................................................................................................13

3.1.1 Jena .............................................................................................................................................13

3.1.2 Redland ......................................................................................................................................15

3.1.3 Sesame .......................................................................................................................................15

3.1.4 KAON .........................................................................................................................................16

3.1.5 rdfDB .........................................................................................................................................16

3.1.6 RDFStore ....................................................................................................................................16

3.1.7 Kowari .......................................................................................................................................17

3.2 RDF Query Languages ......................................................................................................................17

3.3 Document Retrieval Systems ............................................................................................................18

3.3.1 QuizRDF ....................................................................................................................................18

3.3.1.1 Comments on QuizRDF ...............................................................................................20

3.3.2 Semantic Search .......................................................................................................................21
### List of Figures

1. The Semantic Web layer cake.................................................................11
2. The Hypertext Web.................................................................................30
3. The Semantic Web..................................................................................35
4. The Hypertext Web and the Semantic Web............................................37
5. Explicitly specifying the semantics of the document content.....................38
6. Relationships..........................................................................................47
7. SITAR Architecture.................................................................................66
8. SEAL Hyperlinks....................................................................................79
10. Wikipedia Cassius Clay disambiguation page............................................81
List of Tables

1  Results..........................................................................................................................74

2  Wikipedia Results...........................................................................................................82
Acknowledgements

“Guru Sakshat Parabrahman ....,” goes an ancient Sanskrit sloka. Whether or not my Guru, Prof. T.K. Prasad, embodies the infinite, omnipresent, omnipotent and omniscient Brahman is open to debate. The issue falls outside my area of expertise and is beyond the scope of this document. But I can vouch for the fact that he played a major role in my own little universe these last few years and has greatly affected the course of my life. I would like to express my sincere gratitude to Prof. Prasad for his guidance, mentorship, encouragement and support. Without him, I would not have stayed the course.

I would like to take this opportunity to thank Prof. Prabhakar Mateti for his moral support that helped me get through some very rough phases. I also thank him for recommending me to Prof. Prasad and initiating this fruitful relationship.

I would like to thank Prof. Nicholas Bourbakis, Prof. Mateen Rizki, Prof. Arijit Sengupta and Prof. Raj Bhatnagar for taking the time to be on my committee.

All this would not have been possible without the love and support of my family. I want to take this opportunity to thank my parents and sisters for everything. Without their help, I would not have been able to complete even my LKG let alone my PhD.

In school, I was fortunate enough to have teachers who encouraged me to ask questions. I would like to thank my Telugu, English and Physics teachers back in school. They made all the difference.
Last but not the least, I would like to thank all my friends for just being there. Thanks to my Forest Lane friends and roommates who put up with me for so long. Special thanks to my school and college friends who have always been there for me over the years. I would not have been able to get this far without their support.
The Semantic Web initiative is an effort driven by Tim-Berners Lee’s vision, and led by the World Wide Web Consortium (W3C) [WC] to make document content more accessible to machines. The term \textit{World Wide Web} can be seen as a broad label for the group of standards, protocols, languages and other technologies such as HTML, HTTP, etc., whose goal is to enable distributed document sharing. Likewise, the term \textit{Semantic Web} can be viewed as a label for the family of description languages, standards, protocols and other technologies such as RDF [B04], RDFS [BG04], OWL [GH04] etc., which are being added to the \textit{World Wide Web} to enable authoring of content that is more amenable to machine processing. In this sense, the Semantic Web (SW) can be seen as an “extension” of the World Wide Web or the Hypertext Web (HW).

Conceptually, since the Resource Description Framework (RDF) forms the foundation of the Semantic Web, we can visualize it as a labeled graph with resources as nodes and binary predicates as edges (web of data). This is in contrast to the Hypertext Web which is a graph with resources (usually documents) as nodes and hyperlinks as edges (web of documents).

An issue of interest here is the relationship between the two webs: the Semantic Web has been evolving into a property-linked web of RDF data, conceptually divorced from (but physically housed in) the hyperlinked web of HTML documents. Data is being retrieved from the SW using Data Retrieval (DR) techniques and documents are being retrieved from the HW using Information Retrieval (IR) techniques.
Our goal is to view the two webs as a unified whole and devise hybrid techniques to retrieve data and documents from this Unified Web (UW). Taking a unified view will enable us to utilize hyperlink connections from documents to SW data nodes resulting in improved document retrieval in terms of precision, recall and type of documents that can be retrieved. It will also pave the way for keyword-based, hybrid querying of SW data obviating the need for the user to be intimately familiar with the schema information.

1.1 Contributions

This dissertation presents a distinctly unique method of retrieving Web documents and Semantic Web data. It shows how Semantic Web data can be harnessed to enhance Web document retrieval and how information from the Hypertext Web can be exploited to enable easy querying of the RDF graph. Currently, retrieving Semantic Web data and retrieving Web documents are viewed as two completely independent problems. Data is being retrieved from the Semantic Web using Data Retrieval techniques (SQLish query languages) whereas documents are being retrieved from the Hypertext Web using Information Retrieval techniques (keyword based search). This work is the first of its kind to take a combined view of the two webs and uniformly retrieve data and documents from the same unified whole. The Unified Web (UW) model that conceptually unifies the Hypertext Web and the Semantic Web, and the Hybrid Query Language (HQL) that combines Data and Information retrieval paradigms are the core contributions of this dissertation.

The Hybrid Query Language that is based upon the UW model has several traits that set it apart from other query languages. HQL allows keyword based querying of the
RDF graph which enables the user to retrieve data even when schema information is not available. So the user need not know the exact URIs to retrieve RDF data. HQL combines the traditional keyword-based search with graph-based reasoning. This is perhaps best illustrated by HQL’s wordset-pair feature: a user searching for the animal jaguar can convey this to the system using \texttt{<animal>::<jaguar>}.

Another important contribution of this work is the idea of using hyperlinks as semantic annotations. We show how hyperlinks can be viewed as semantic markup and how this can be used to annotate legacy documents thereby improving document retrieval.

1.2 Outline

This document is organized as follows. We first provide a brief introduction to the Semantic Web in Chapter 2. In Chapter 3, we provide a brief survey of related research. In Chapter 4, we motivate the work and elucidate our goals. In Chapter 5, we present the Unified Web model, and in Chapter 6, we present the Hybrid Query Language (HQL) for the Unified Web. In Chapter 7, we describe our system SITAR and evaluation details and conclude with Chapter 8.

1.3 Conventions

The following are some of the conventions and terms used in this document.
1.3.1 Hypertext Web and the Semantic Web

In order to avoid confusion, we will refer to the current World Wide Web as the “Hypertext Web” or “HTML web” or simply “the Web” (with a capital W). Likewise, we will refer to the Semantic Web as SW.

1.3.2 URI vs. URL

A resource conceptually maps to an entity [BFS03]. Anything that has an identity can be considered a resource. Examples of resources are persons, telephone numbers, web pages, and other abstract entities. URIs are the most general set of identifiers used to identify resources. URIs include Uniform Resource Locators (URLs) and Uniform Resource Names (URNs). A URL is something that explicitly tells us where the resource is and how to access it. For example, the URL `ftp://ftp.wright.edu/abc.txt` tells us that the `abc.txt` can be obtained via file transfer protocol from Wright State University’s FTP server. Therefore, a URL identifies a resource based upon its location and access mechanism. A URN on the other hand is an identifier that is independent of the location of the resource. An example of URN is `urn:isbn: 0312278497`. This URN is the ISBN number of a book. The URN must persist and remain unique even if the resource is physically unavailable. We will use the term “URI” (which is the current convention) to refer to resources and will use “URL” only when we want to explicitly point out that the resource is a web page. A good discussion of URIs, URLs and URNs can be found in [BFS03].
1.3.3 Formal Query vs. Keyword Query

Any query language that has been rigorously defined can be considered a “formal” query language. Technically, this includes keyword-based query languages that systems such as Google offer users. In our discussion, we will use the term “formal” to refer to query languages such as SQL which allow the user to unambiguously express an information need to the system. This is to differentiate them from keyword-based query languages which inherit the ambiguity that is inherent in natural language.
2 The Semantic Web

The World Wide Web (WWW) is primarily a web of documents. The content of the documents is usually natural language text whose semantics is mostly inaccessible to machines. This text is typically marked up using languages like Hypertext Markup Language (HTML) [HTML]. The markup basically describes the structure of the document. That is, the markup describes which part of the document is a heading, a paragraph, a hyperlink and so on. Web browsers can use this information to display the documents appropriately to humans. Therefore, the primary purpose of markup is to make explicit the logical structure of a document and leave the presentation details to the browser.

HTML markup helps the machines recognize different components of a document. But it does not help the machines “understand” the content of these components and hence the document. In other words, HTML does not make the semantics or meaning of the document content accessible to the machines. In this sense, documents marked up by HTML are no different from those that are not marked up. We can refer to such documents as natural language documents or free-text documents or perhaps more controversially unstructured data.

The WWW has revolutionized the world. It is undoubtedly a huge success and has seen tremendous growth in the recent past. The underlying technology is simple enough for a lay user to create and add documents to this repository. This has resulted in an explosion of information. As a result, finding and using relevant, precise, timely and reliable information on the Web is becoming more and more difficult. Search engines like
Google [Goo, BP98] and Yahoo! [Yah] help the users search for documents on the Web using keyword based queries. The problem with the search engines is that they either retrieve too many documents (high recall, low precision) or too few documents (low recall) [AV04]. The user has to wade through the returned documents looking for necessary information. Also, search engines are not content aggregators. If the information that the user is looking for is distributed over several pages, then the user has to extract pieces of information from different documents and put them together. For example, consider the case of a Ph.D. student with the information need “Find all the technical papers on Semantic Web technologies published in the last five years.” Using existing search engines (and keyword queries) to fulfill such an information need is not an easy task even though all the information to satisfy the need is (arguably) present on the Web.

The problem is that the web documents are in natural language and the semantics of their content is not accessible to machines. The underlying technology of current search systems tries to guess the meaning of the query and of the documents. Typically, it does so by examining the syntactic content of the documents (term frequency, inverse document frequency etc.). If all the web content were to somehow be translated into appropriate formal logic such as First Order Predicate Logic (FOPL) which is more amenable to machine processing, then we can build systems that can search for documents based upon their semantic content rather than syntactic content. Automated agents can reason with the information available on the web and this can open doors to endless possibilities [H01]. NLP is a branch of AI that has been trying to do just that. NLP is about automatically “translating” natural language text into representations that
are more amenable to machine processing. But unfortunately, the task appears to be harder than researchers had originally imagined. One alternative is to manually encode the web content in machine-processable form. But it is impractical and the trade-off between the effort required and the expected benefits is probably not worth it. The solution to this would be to describe rather than translate web documents (or parts thereof) using formal languages. These (formal language) descriptions of web documents (or resources) can be created either manually or automatically.

The Semantic Web [SWA, BHL01] initiative led by World Wide Web Consortium (W3C) [WC] aims at making the content of the web more accessible to machines. The basic underlying data model of the Semantic Web (SW) is the Resource Description Framework (RDF) [B04]. RDF is an abstract data model expressed using XML. The basic concepts of RDF are resources, properties and statements. A resource is anything that can be identified by a Uniform Resource Identifier (URI) [BFM98, UIG01]. A property is a resource that describes relations between resources. RDF describes resources by making statements about them. Each statement is a triple of the form 

<subject, predicate, object>. The subject and the predicate (property) of the triple can only be a resource whereas the object of the triple can either be a resource or a literal. Literals, for all practical purposes, are nothing but strings. For example, the statement 

John Smith is the son of Mary Smith

can be represented as the following RDF statement.

< mailto:jsmith@abc.com
    http://www.relationsOnto.com/sonof
    mailto:msmith@xyz.com >

Here, the subject is John Smith (identified by the URI mailto:jsmith@abc.com). The object of the statement is Mary Smith (identified by the URI mailto:msmith@xyz.com).
The URI http://www.relationsOnto.com/sonof is a property that connects the two persons. The RDF document that encodes this statement (using XML syntax) looks something like this:

```xml
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:mydomain= "http://www.relationsOnto.com/">
  <rdf:Description rdf:about="mailto:jsmith@abc.com">
    <mydomain:sonof rdf:resource="mailto:msmith@xyz.com" />
  </rdf:Description>
</rdf:RDF>
```

The statement can also be represented using a directed graph as follows.

![Directed Graph](image)

In the above graph, http://www.relationsOnto.com/sonof has been shortened to sonof for the sake of brevity. We will follow this convention throughout this document whenever the domain name of the property is not important.

The use of URIs removes the problem of ambiguity that natural language suffers from. The URI mailto:jsmith@abc.com refers to only one entity. Note that at this point, a machine processing the above statement would know nothing about the entity represented by the URI. It does not know that the URI represents a person and that the person’s name is John Smith etc. RDF is a general, domain-independent data model for representing information. Users can use their own vocabularies and make statements to describe resources. But for the machines to do any meaningful processing of these statements, the user has to define the semantics of the domain by describing the domain vocabulary. In other words, the user has to create domain ontology. Ontology can be defined as “an
explicit specification of a conceptualization” [G93]. The domain ontology can be created using RDF Schema (RDFS) [BG04] (itself described using RDF). For example, the user can define a class named person and make mailto:jsmith@abc.com and mailto:msmith@xyz.com instances of that class as follows.

```xml
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:mydomain= "http://www.relationsOnto.com/">
   <rdf:Description rdf:ID="Person">
     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
     <rdfs:subClassOf
       rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
   </rdf:Description>

   <rdf:Description rdf:about="mailto:jsmith@abc.com">
     <rdf:type rdf:resource="Person"/>
     <mydomain:sonof rdf:resource="mailto:msmith@xyz.com"/>
   </rdf:Description>

   <rdf:Description rdf:about="mailto:msmith@xyz.com">
     <rdf:type rdf:resource="Person"/>
   </rdf:Description>
</rdf:RDF>
```

In a similar fashion, the user can also define properties and specify restrictions on the properties. The sonof property in our example can be defined in such a way that its domain and range would be restricted to instances of person. This example is intended to give an idea as to what the Semantic Web is all about. A complete description of RDFS is out of the scope of this document.

Some of the expressivity issues that arose with RDFS [AV04] led to the development of Web Ontology Language (OWL) [GH04]. OWL is derived from DAML+OIL Web Ontology Language. OWL is a W3C recommendation and is the standard language for creating SW ontologies. OWL comes in three flavors each one with a varying degree of expressivity – OWL Full, OWL DL (DL stands for Description
Logic) and OWL Lite. RDFS and OWL (DL and Lite) can be viewed as specializations of predicate logic [AV04]. A full treatment of all these languages and the expressivity issues involved is out of the scope of this document.

Once the domain ontology is defined, an automated agent can use the ontology information (and inference rules) to reason over the data and infer additional information. In our example, let us suppose that there is another person named Peter Smith represented by the URI mailto:psmith@xyz.com and that the following triple exists in the database.

\[ \text{mailto:psmith@xyz.com} \ \text{husbandOf} \ \text{mailto:msmith@xyz.com} \]

Then we can write a rule that essentially says that if person A is the husband of person B, and if person C is the son of person B, then person C is also the son of person A. The agent can then use this rule to infer the following triple.

\[ \text{mailto:jsmith@abc.com} \ \text{sonOf} \ \text{mailto:psmith@xyz.com} \]
Figure 1 lays out the Semantic Web vision as a layered cake where each layer is built on top of the other. RDF model is built upon the idea of URIs and it has XML syntax. RDF Schema and OWL are ontology languages which themselves are described using RDF. These languages map to well-understood formal models. Automated agents can use rules and ontological information to reason with the data and infer new information. The proof layer represents the fact that an agent must be able to justify how it has arrived at a given fact by keeping track of reasoning traces. It specifies proof representation and proof validation. The SW is like a huge database of facts. But the nature of the web is such that anybody can add new facts to this database. It is important for automated agents to be able to determine whether a fact is to be trusted or not. The topmost layer of trust (built upon encryption and signature) signifies this.
3 Related Research

Our research is about how IR and DR techniques can be combined to devise hybrid techniques to retrieve documents and data from the Unified Web which is an encoding of the SW and the HW. It draws from the research that has been performed in Information Retrieval, Databases and AI (Knowledge representation and reasoning). There have been many RDF storage systems that adopt a database approach to retrieve data from the SW using formal languages. There have been many Information Retrieval systems retrieving documents from the Web. There have been some hybrid systems that try to combine these two paradigms to perform document retrieval. In this chapter, we give a brief overview of some of the existing systems.

3.1 RDF Storage Systems

A survey of open source RDF storage systems can be found in [B02b]. A survey of RDF storage systems that use a database to store RDF data can be found in [B03]. Here, we provide a brief description of some of the storage systems.

3.1.1 Jena

Jena is a (rather popular) Java framework for storing and querying RDF data [Jen, M02, WSKR03]. RDF data can be stored either in a database or in-memory. The components of the framework include APIs for processing RDF (reading, writing, navigating the graph, querying etc.), API for processing ontologies in RDFS, OWL and DAML (creating an ontology model over which reasoning can be done), support for querying
using RDQL [S04] and SPARQL [PS06] etc. We now give a brief description of the storage schema of the system to give an idea as to how RDF storage systems store data.

The first version of Jena (Jena1) used two different database schemas- one for relational databases and a special one for Berkeley DB (BDB) [WSKR03]. The schema for relational databases consisted of a statement table, a resources table and a literals table. Every resource and literal was given a unique Id. Resources were stored in the resources table (ID, Value) and the literals were stored in literals table (ID, Value). Statements were stored in the statement table (subject, predicate, ObjectURI and ObjectLiteral). To distinguish object literals from object URIs, two columns were used for the object of the statement. As is obvious, every find operation would require multiple joins between the statement table and the literals table. The schema for BDB [BDB] stored all parts of the statement in a single row. But each statement was stored three times indexed by subject, predicate and object respectively, thereby increasing storage cost. It was found that data was accessed much faster from BDB suggesting that denormalized relational schema might reduce response times.

Jena2 uses a denormalized schema in which resource URIs and literals are stored directly in the statement table. The statement table has just three columns (subject, predicate object). In order to distinguish object URIs from object literals, prefixes are used. The schema still has a resource table and literals table but only long literals (whose length exceeded a certain threshold) and URIs are stored in these tables. In order to compensate for the increased storage cost, several optimizations are performed (for example, common URI prefixes are compressed). Jena2 also has an additional
distinguishing feature. It allows the creation of separate property tables based on arbitrary properties. A property table stores all the pairs related by the property.

3.1.2 Redland

Redland [Red, B01] is another RDF application framework like Jena that allows RDF data to be stored, queried and manipulated in C, Perl, Python, TCL and other languages. Like Jena, it provides APIs for manipulating and querying RDF. RDF graphs can be stored in-memory or in a database (BDB, MySQL etc.) It provides support for querying using RDQL and SPARQL. Redland stores RDF statements (in-memory or persistent storage) using hashes. A hash is the mapping from a key to a value with duplicates allowed. Statements are stored using three hashes corresponding to the following key-value pairs: (Subject & Predicate, Object), (Predicate & Object, Subject) and (Subject & Object, Predicate).

3.1.3 Sesame

Sesame [Ses, BKV02] is a Java framework for storing and querying RDF. It can be deployed on relational databases, in-memory, file systems, etc and supports both local and remote access. It supports querying using several different query languages including SeRQL and SPARQL. Like other systems, it provides APIs to manipulate RDF graphs. Sesame supports RDF Schema reasoning, and plug-ins for OWL support are available. The native Sesame store uses B-Trees for indexing statements where the index key consists of three fields – subject, predicate, and object.
3.1.4 KAON

KAON [Kao, VOSM03] is a Java based suite of tools that focuses on scalability and efficient reasoning with ontologies. The ontology language that is supported is based on RDFS with proprietary extensions. The infrastructure targets business applications and gives importance to security, consistency, extensibility etc. API for manipulating RDF with support for both in-memory and persistent storage (any SQL2 compatible database) is provided. KAON2 [Kao2], a successor to KAON supports OWL-DL.

3.1.5 rdfDB

rdfDB [Guh] is a Sleepycat BDB based database for RDF. It supports C and Perl access. The database can be queried using a high level SQLish query language provided. It supports RDFS along with some basic inferencing. The system uses three Sleepycat B-Tree databases to index each statement using the following key value pairs (Subject&Predicate, Object), (Object&Predicate, Subject) and (Predicate, Subject).

3.1.6 RDFStore

RDFStore [Rst, RVB03] is a Perl/C toolkit to process RDF. Data can be stored and retrieved from in-memory hashed storage or from the local disk using BDB. Querying can be done using RDQL query language. The programming API supports contexts, reification, multi-lingual literals, etc.
3.1.7 Kowari

Kowari [Kow, AGW05] is a transaction-safe storage infrastructure for RDF data. Kowari supports iTQL and RDQL query languages. Kowari stores RDF in its native transactional data store. The system is designed such that other data stores can be plugged into it. The native data store consists of a Statement Store and a String Pool. Statements are stored as a quad of nodes consisting of the subject, predicate, object and meta nodes. Each node is assigned 64-bit integers and statements are therefore stored as a quad of integers. The URIs and literals are stored in the String Pool. The String Pool is built using AVL trees whereas the statement store is built using a hybrid of AVL and B*-Trees. The system uses Lucene[Luc] to support full-text search of literals.

RDFSuite [For, AKC+01], 3store [HG03], BRAHMS [JK05], YARS [HD05], Parka [Par02], Haystack [HKQ02] are some of the other data stores found in the literature but the list is by no means exhaustive.

3.2 RDF Query Languages

SPARQL has become a W3C candidate recommendation as recently as April 2006. As is evident from the brief description of the storage systems above, many RDF query languages have been designed and implemented. An excellent, comprehensive survey of RDF and XML query languages is provided by Bailey et al. [BBFS05], and Haase et al. [HBEV04], make an in-depth comparison of six RDF query languages.
3.3 Document Retrieval Systems

The previous sections described RDF data stores and formal RDF query languages. What is of interest to us is how the RDF data from the SW can be utilized to retrieve documents from the Web. The following is a brief overview of systems that try to address this issue. We give an in-depth view of some systems that are representative of the work being done in this area and briefly mention other relevant systems. At this point, it is also worth mentioning that our work also focuses on how information from the Web (documents and hyperlink information) can be used to improve RDF retrieval from the SW.

3.3.1 QuizRDF

QuizRDF, in the words of the authors [DWK02], is an “information-seeking system….which combines traditional keyword querying of WWW resources with the ability to browse and query RDF annotations of those resources.” Its “…index allows both keyword querying against the full text of the document and the literal values occurring in the RDF annotations, along with the ability to browse and query the ontology.” QuizRDF views every node as a document-node and there are no non-document nodes in the system. The URL of the document node is an instance of some class (say Employee) and this can be written as follows.

\[ <URL_{n}, \text{type-of, Employee}> \]

These nodes (RDF resources) are annotated with what are called “content descriptors”. The descriptors (essentially words and phrases) are obtained from the body of the document. They are also obtained from the literals that are related to the node by a
property. For example, if Employee class has a property named last_name, then the value of the property can be written as follows.

\[<URL_n, \text{last}_\text{name}, \text{“Miller”}>\]

The descriptor information is maintained by the system as follows. If the descriptor (say “George Miller”) is obtained from the body of the document then the system keeps track of this information by creating triples of the following type (lets call this type 1).

\[< \text{“George Miller”}, \text{Employee}, \emptyset > \rightarrow URL_n\]

Likewise, if the descriptor is obtained from the literal related to the node by a property then the system keeps track of the information by creating a triple of the form (type 2):

\[<\text{literal, class, property}> \rightarrow URL\]

In the above example this would translate into:

\[< \text{“Miller”}, \text{Employee, last}_\text{name}> \rightarrow URL_n\]

QuizRDF provides a user interface to query and browse the results. When the user enters a free text query, the system retrieves URLs that contain the words (presumably) in their body (and hence using type 1 triples). Recall that each of these URLs is an instance of some class. The system then allows the user to filter these results (by using a drop-down menu) based upon their class types. If the user chooses, say, the Employee type, the system then allows the user to search (using keywords) the resulting set based upon the content of their property literals (type 2 triples). That is, the user can now search the resulting “Employee” documents based upon the content of their “last_name” literals. The system also displays the names of the URLs that are related to the node by a property. For example, if the “Employee” class is related to the “Skills” class by the property “HasSkills”, then the system displays the name “Skills”. The user can then
perform searches based upon “Skills” by clicking on the name. The system has a
graphical user interface which provides the users with a simple mechanism to query and
explore the information.

3.3.1.1 Comments on QuizRDF

There are many differences between our work and QuizRDF. To begin with, our model is
much more general compared to QuizRDF’s model and seeks to unify the data-centric
Semantic Web and document-centric Hypertext Web. The nodes in our system need not
have a document content associated with them. A node which has a document associated
with it, need not be instance of any class. A node in our system is indexed not just by
words but also by URIs and the triples that it asserts. Also note that in our system, the
nodes are indexed by what we call “URI index words” which serve to describe the URI of
the node. The words in this set may come from the literals related to the node by
properties such as rdf:label and rdf:name but not from properties such as “last_name” or
“sonOf”. The user is expected to compose triple queries to retrieve such information.
Ours is a meta-model - the document asserting the triple is also indexed by the system
and should be searchable.

Our query language of the system allows the users to query both for documents
and the data (nodes with no documents attached to them). It allows the user to take the
underlying structure of the data into account while composing queries even while
searching for documents. Last but not the least, our system allows the users to compose
queries around more than one class (which the authors explicitly say is a drawback of
their system).
3.3.2 Semantic Search

Semantic Search [GMM03] aims at improving traditional web search. The authors view the HW and the SW as two separate but connected entities. The system does not make use of any semantic markup that can be found in a HTML document. The robots are expected to gather the markup and add it to the SW. The aim of the system is to improve searching of documents on the HW by making use of the data found on the SW.

The authors distinguish between Navigational searches (the user is looking for a document which contains the search words) and Research searches (the user is trying to find out information about an entity denoted by the search terms). The system attempts to improve the quality of Research searches in the following two ways. The first way is by augmenting the traditional search results with relevant data from the Semantic Web. The second way is by zeroing on the intended denotation of the (potentially ambiguous) search phrase.

In order to augment the search results with the SW data, “relevant data” must first be retrieved from the SW. The system makes use of the TAP infrastructure which provides a query interface to the Semantic Web. Data can be retrieved from the SW by using one of the following three interfaces. GetData interface allows the SW to be queried for property values. It is of the form:

\[\text{GetData(<resource>, <property>) => <value>}\]

Resources are “typically” referred to using their URIs. The next interface is called the Search interface and takes a string and returns resources (from named graphs). The keyword to resource mapping is done, apparently, by examining the title and some
properties of the resource. In addition, there is an interface called the reflection interface which returns a list of incoming and outgoing links of a given node thereby facilitating the exploration of its vicinity.

Given a search query, the system sends it to a search engine (Google) to retrieve relevant documents. It also uses the query (and TAP infrastructure) to retrieve relevant data from the SW. The documents and the data are then displayed side by side to the user. The system also helps the user to communicate the intended denotation of the (potentially ambiguous) search term to their system. It does so by taking the search term and listing all the corresponding URIs and letting the user choose the intended denotation. Once the intended denotation has been identified, the system has to then determine which of the documents in the database talk about this particular sense of the term. This part of their work is still in progress.

3.3.2.1 Comments on Semantic Search

Just like the Semantic Search system and TAP we use a light weight query language to retrieve data from the SW. But, as will be seen, the query language that we are proposing is more expressive (word sets, word-set pairs, triplets etc).

As far as document retrieval is concerned, unlike Semantic Search, our model does not view the SW and the HW as two separate webs. The primary focus of Semantic Search is to help users with their research queries. So, it is not clear how the system can be used to, say, search for SW documents like ontology documents. Semantic Search treats the HTML repository as a separate repository to be queried using keyword based queries (it uses Google to query the repository). We believe that a HTML document can
be annotated with semantic markup and this markup can be used to improve retrieval performance. We also believe that the same underlying framework can be used to retrieve both the data and the documents.

### 3.3.3 OWLIR

OWLIR [MF03, SFJ+02] is a system designed to retrieve both free-text and semantic markup in RDF, DAML+OIL or OWL. What is especially relevant about OWLIR is that it treats each distinct RDF Triple in the document as an indexing term. The authors advocate that search and inference should be tightly bound and suggest using semantic markup as indexing terms for IR systems. It is also worth noting that the query to the system is expressed as a document consisting of triples (exact URIs) and free text.

#### 3.3.3.1 Comments on OWLIR

OWLIR’s approach of treating a triple (that appears in the document) as an indexing term closely corresponds to what we are proposing. Like OWLIR, we want to provide a hybrid query language that has both formal and keyword components.

But, as will be seen later, one of the goals of our system is to allow the users to formulate even the formal queries using keywords. For example, in our system, a user can submit a semi-formal query of the form (?x son Mary) to retrieve URIs/triples and documents from the database. This involves finding bindings to the variable ?x. It is not clear if OWLIR tries to find such bindings – it seems to be literally treating triples as indexing terms and doing “verbatim” triple matches to retrieve documents. OWLIR is focused on document searches whereas we are also interested in supporting graph
searches. Also, note that there is an element of uncertainty even in the formal component of the query language that we are proposing (because of keywords). Therefore, ranking the retrieved results becomes important.

### 3.3.4 Swoogle

Swoogle [Swoo, D+05a] is a retrieval system for the Semantic Web. Swoogle aims at finding, indexing and retrieving Semantic Web documents (SWDs). An SWD is defined as “a web page that serializes an RDF graph using one of the recommended RDF syntax languages such as RDF/XML . . .”[D+05b]. In addition, Swoogle also aims at helping users find Semantic Web terms (URIs). What makes Swoogle especially relevant to the work that we are doing is the fact that Swoogle allows users to search for SWDs and URIs using keyword based query language.

Note that most of the current SWDs available are created by researchers and many of them are ontologies. In order to search for SWDs, Swoogle maintains three types of metadata for a document. They are document metadata, content metadata and relation metadata.

The document metadata consists of information such as the URL of the SWD, its file extension, document size etc. Content metadata describes the RDF graph encoded in a SWD. The graph is analyzed to see how many triples in it are defining a new class and how many of them are instantiating existing classes. The ratio between the two measures (ontology ratio) is then used to heuristically determine to what degree the document can be considered an ontology. Swoogle uses the information found in the graph (RDF/XML fragment) of an SWD to index the SWD and later retrieve it. The triples of the graph are
analyzed and the URIs used to (presumably) index the document are carefully (and heuristically) chosen so as to improve retrieval precision. For example, if a triple has the property `rdfs:comments` then the URI of the property itself does not tell us anything about the SWD. So, it is not used to index the document. All of this information is (again presumably) encoded and stored as content metadata.

Relational metadata characterizes the binary relationships between SWDs and XML namespaces, SWDs and RDF resources, and SWDs and other SWDs. If an SWD is using a particular namespace, then this information is captured and stored. This would enable the users to search for SWDs based upon the namespace that they are using. An SWD can have the following relationships with an RDF resource. It can define the resource (`define-class` if the resource is a class and `define-property` if the resource is a property), it can use the resource (`use-class`, `use-property`) or it can populate or instantiate the resource (`populate-class`, `populate-property`). RDFS allows two SWDs to be linked using `rdfs:seeAlso` and `rdfs:isDefinedBy` properties. Likewise, OWL also allows association of SWDs by properties such as `owl:priorVersion` and `owl:backwardCompatibleWith`. All this information is (presumably) captured and stored as relational metadata and used while answering queries.

Swoogle allows the users to search for terms (URIs) using keywords. It uses a variety of techniques to obtain the index words of a URI. For example, the words are obtained by analyzing the URI itself and by examining the local names of URIs. They are also obtained by analyzing the literals that appear in the triples defining the resource (for example, `rdf:comment`). Swoogle allows users to query for terms based upon the ontology language used to define the terms. Currently, four “families” of terms are supported –
OWL, RDFS, RDF and DAML. The user can also query for terms based upon their type i.e., whether the term has been defined as a class or a property or both.

Swoogle retrieves SWDs and terms in response to a keyword-query. In addition, it also ranks the retrieved SWDs and terms. The ranking algorithms are inspired by Google’s PageRank [PBMW98] algorithm.

Other similar ontology search systems are OntoKhoj [PSL03] and OntoSearch [ZVS04] and AKTiveRank [AB05].

3.3.4.1 Comments on Swoogle

Swoogle allows the users to search for SWDs and URIs using keyword based queries. An SWD, by definition has a fragment of semantic description (example, RDF/XML). Our aim is to use the existing hyperlinks as semantic annotations while retrieving documents. So, every document with a hyperlink is a potential SWD in our work. Also, Swoogle does not allow users to search for sub-graphs in the RDF graph. In other words, queries return SWDs (documents) and URIs as opposed to triples and graphs. In our system, we want the users to query for triples using a keyword-based query language.

3.3.5 Others

The system described in [RSA04] is similar to QuizRDF in the sense that a document is viewed as an instance of a concept and annotated with the literals of the properties related to the concept and also by the content of the document. Their search architecture combines classical search techniques with spread activation techniques so that related concepts (and hence documents) can be retrieved given a query.
[VFC05] describes an ontology-based scheme for semi-automatic annotation of documents and a retrieval system that uses the annotations to retrieve documents. Their model is based upon the adaptation of the vector-space IR model where the keyword-indices are replaced by an ontology-based KB. They also describe a weighting-scheme and a ranking algorithm for the retrieved documents.

[ZZY+05] seeks to combine IR with formal query and reasoning to enhance search in semantic portals. The IR model of their choice is the fuzzy DL IR model. They try to combine it with and a formal DL query to retrieve documents.

### 3.4 Summary

Storing and retrieving RDF data is an area of research that has been well explored by researchers in the recent past [B02b, B03, BBF05, HBEV04]. Retrieving RDF data is typically viewed as a data retrieval problem and, not surprisingly, most of the query languages have the SQL flavor [BBF05]. When seen purely from the perspective of querying the RDF data, our Hybrid Query Language (HQL) is unique because, being key-word based, it allows the users to explore the RDF graph even without any knowledge about the underlying schema (namespaces, exact URIs, ontologies, etc.). The user can use HQL to quickly get a feel for the underlying data. HQL has many features such as wordset pairs which are novel.

As far as document retrieval is concerned, there are several retrieval systems that retrieve documents based upon their annotations(descriptions) [GMM03, DWK02, MF03, D+05b, RSA04, ZZY+05, VFC05], but none seems to aim at retrieving HTML, RDF and hybrid documents (that is, all the three types). We index a document based upon words,
URIs, and triples that can be extracted from the document and give the user a light-weight query language to retrieve documents based upon this information. The query language is hybrid in the sense that it has both “formal” and keyword components but what is unique is that the “formal” component itself is expressible using keywords.

Unlike our unified approach, Semantic Search [GMM03] treats the SW and the HW as two separate repositories and aims at retrieving documents from the HW (in fact they use Google to search for documents). It lets the user communicate the disambiguation information using the user interface. Like our system, SITAR, quizRDF[DWK02] indexes a document (URL) using words obtained from its body as well as from the literals of triples in which its URL participates. Like Semantic Search, quizRDF too uses a GUI to let the user communicate disambiguation information.

SITAR views the SW and the HW as a unified whole (unlike Semantic Search). One benefit of this, compared to quizRDF, is that the URL of a document is also indexed by the anchor text words. Further, SITAR indexes a document using any URIs (linksTo) or triples (asserts) that can be extracted from the document. This allows it to index and retrieve RDF documents (and hybrid RDFa [AB06] kind of documents in the future). Also, unlike the above two systems, the user can specify the disambiguation information (the “class”) using word-set pairs, and use it in conjunction with linksTo information to retrieve documents.

Swoogle[D+05b] specializes in retrieving ontology documents and URIs. It doesn’t seem to index HTML documents or support triple search or keyword-based querying of the RDF graph.
OWLIR’s[MF03] approach of treating a triple (that appears in the document) as an indexing term corresponds to what we are doing. But the way the indexing information is used and the nature of the query language is quite different. HQL is keyword-based and so the users can retrieve an “asserting” document even when the exact URIs are not known. Also, we index a document based upon the component URIs of the triples and the hyperlinks that appear in the document (linksTo).

There are several other systems [RSA04, ZYZ+05, VFC05] that perform hybrid retrieval but our system is different due to the reasons discussed above and due to the fact that we view SW and HW as a single UW. We situate the SW data and the HW documents side by side and query the Unified Web using HQL which has both keyword and “formal” components. We also exploit existing hyperlink (linksTo) connections between HW documents and SW nodes while retrieving documents. This idea of viewing a hyperlink as semantic annotation (or propagating semantics through hyperlinks) is something very unique.

There are many systems that perform keyword to URI mapping. The keywords used to index the URI usually come from the URI itself or from property literals that the URI participates in. The query language that we are proposing involves mapping keywords to URIs. What makes our approach unique is that we believe that the index terms of a URI can also come from documents.
4 Towards the Unified Web

In this chapter we discuss the current methods used to retrieve documents from the WWW and data from the Semantic Web. We discuss the limitations of these methods and motivate the need for Unified Web.

4.1 Retrieval from the Hypertext Web

The World Wide Web is a large, heterogeneous collection of documents and other objects connected by hyperlinks (Figure 2).

![Figure 2: The Hypertext web](http://www.cs.wright.edu/~barnes.html)

Most of the content on the World Wide Web is in the form of free-text and is largely inaccessible to machines due to the limitations of NLP. In this vast web of mostly unstructured data, search systems like Google, Yahoo etc., try to help users find information. Such systems typically allow the user to communicate her information need to the system using a simple keyword based query mechanism. Given a keyword query, such systems typically return a set of documents that are likely to satisfy the users’
information need as represented by the query. Such systems are called Information
Retrieval (IR) systems and are distinguished from traditional database systems or Data
Retrieval (DR) systems mainly by the need to interpret the user query and the contents of
the repository. This need arises mainly because of the inherent ambiguity in natural
language which renders both the query and the contents of the web documents
semantically ambiguous. Also, there is ambiguity regarding the user information need
itself – the user may not know what she is looking for or may have incomplete
information while formulating the query. Because of the gap between the information
need and the semantics of the query and the documents, an IR system conceptually ranks
all the documents in the repository based upon their likelihood of satisfying the user’s
information need and returns the top n documents from the ranked list (discarding the
bottom N-n documents). The system is said to “retrieve” those n documents.

In order to illustrate the search process on the web, consider the case of a lay
person with the question “What is the name of John Smith’s wife?” She can express this
information need to the system by submitting the keyword query {John Smith wife
name}. Given such a query, the current search systems typically return hundreds of
documents that talk about Johns and Smiths and several different John Smiths among
other things. The user can wade through the list of documents and eventually find the
information she is looking for. But such a task is usually overwhelming. After examining
some of the documents returned by the system, perhaps the user realizes that there are
several different John Smiths on the web and that she has to indicate to the system which
particular John Smith she is interested in. The user also realizes that she has to somehow
convey to the system that John and Smith are not two different people. So she modifies
the query to \{“John Smith” actor wife\} thereby indicating to the system that she is only interested in John Smith the actor and not anybody else. Also, note that the user has now realized that the word name is not necessary and is implicit in the query (in fact it adversely affects the quality of the results). The problem now is that there could be several John Smiths in the acting profession. Also, the system may now retrieve pages about actors who are perhaps being interviewed by an individual named John Smith who is asking the actor a question about his wife. The hope is that the system will interpret the query appropriately and guess what the user’s information need is and that the system’s relevance ranking algorithm will ensure that the user will find what she is looking for in the top few documents. Perhaps after examining the results, the user will change her query to \{“John Smith” actor biography “Needle in a Haystack”\} thereby indicating to the system that she is looking for information on John Smith who, perhaps, is well known for his performance in the movie “Needle in a Haystack”.

The above example illustrates the travails that one has to endure to find information on the web because of its unstructured nature. But it also illustrates the fact that the keyword based query mechanism is simple and powerful enough to allow even lay users find information on the web with relative ease. It allows the users to probe the repository and get an idea as to what kind of information is present in it. This can then help the users form more effective queries. So, if the documents being returned by the system are not directly fulfilling the user information need, they are helping her understand the nature of the repository and the nature of the system itself which in turn leads to query refinement/modification.
An ideal search system for the web would, perhaps, be a question answering system which would take a natural language query such as “What is the meaning of Life?” from the user and gather relevant information from various sources on the web, digest the information, and return a clear cut answer such as “According to h2g2.com, the meaning of life is 42 but apparently plato.stanford.edu does not quite agree with this view......and after reviewing all this information, I conclude that the meaning of life is.......” But the current search systems are not question answering systems (though Ask [Ask] tries to be one). These systems aim at pointing the user to locations on the web where the answer to their questions can probably be found. These systems typically return a list of documents. The user then has to explore each one of them and glean information either from the document itself or from other documents in its vicinity to find answers to her questions. In other words, the user has to explore and extract information from the documents. Going back to our John Smith example, when the user submits the query {“John Smith” actor biography “Needle in a Haystack”} perhaps the first document returned by the search engine has the following content:

“John Smith, the actor who is well known for his performance in the movie Needle in a Haystack is currently working on the movie The Cornucopian Nightmare ...

John Smith’s biography can be found here.”

In this page, here is the anchor text of a hyperlink to another webpage (say http://www.imdb.com/js.htm) which will actually contain John Smith’s biography and hopefully the name of his wife. So typically, the system returns a list of documents each one of which is a point from where the user can start browsing and hopefully find answers to the questions. Therefore, searching on the web using the current retrieval
systems can be thought to be a series of “Query and Explore” steps laced with query refinements/modifications.

4.1.1 Limitations

In the above example, the user has a specific information need and is conducting what is known as a “research search” [GMM03]. The keyword based query language is such that the user cannot unambiguously convey to the system that she is looking for information about John Smith the actor. Even if the query language were to provide such a mechanism to the user, the system would not know how to process the query because of the inherent ambiguity in the natural language documents. The system then has to deal with problems like polysemy (one term, different meanings/entities) and synonymy (one entity, different names/terms) determine which of the documents are about John Smith the actor. The end result of all this is poor precision. Recall is typically not a problem unless synonyms (aliases, co-references etc.) enter the picture – the document that contains information about John Smith’s wife is likely to contain the terms “John” or atleast “Smith”.

4.2 Retrieval from the Semantic Web

As explained in the introduction, the Semantic Web is a “web of data” built upon the Resource Description Framework (RDF). The fundamental component of RDF is the RDF triple or the RDF statement. Recall that the RDF Triple has a subject, a property (also known as predicate) and an object. The subject and the predicate can be resources (anything identified by a URI) whereas the object can be either a resource or a literal
Thus the entire SW is encoded in RDF triples and can be viewed as one huge formal database of RDF triples. Therefore, the problem of retrieving information from the SW can naturally be viewed as a Data Retrieval (DR) problem. Not surprisingly, database techniques have been extensively applied to the problem of storing and retrieving RDF data. There are several RDF databases in the literature (see Chapter 3). There have also been many formal RDF query languages developed to query such databases (see Chapter 3). Most of these query languages resemble SQL. SPARQL [PS06] is an RDF query language that is a W3C Candidate Recommendation. The following examples shed more light on SPARQL.

The Semantic Web can be conceptually visualized as a graph whose nodes are resources connected by edges labeled with properties (Figure 3).

Figure 3: The Semantic Web
The above example is about a resource (individual) identified by the URI mailto:bsmith@cs.wright.edu. The individual’s first name is “William” (literals are shown in rectangular boxes) and last name is “Smith”. Our William Smith is, apparently, a university professor and teaches the course identified by the URI http://www.cs.wright.edu/CS420. Also note that William Smith has a homepage identified by the URI http://www.cs.wright.edu/~bsmith. In order to find out what courses William Smith is teaching, one can compose the following SPARQL query.

```sparql
PREFIX univ: <http://www.univ.com/>
SELECT ?course
WHERE
{
  <mailto:bsmith@cs.wright.edu> <univ:teaches> ?course .
}
```

The system should examine the stored triples and find the right bindings for the variable ?course and retrieve them as the answer set.

One of the motivating factors behind the SW initiative was the idea of making web content accessible to machines thereby enabling document search based upon semantics as opposed to syntax. But the SW has been evolving into a separate world of its own parallel to the Hypertext web. This parallel web, however, is connected to the Hypertext web at some points (See Fig 4).

In Figure 4, the three web pages in the Hypertext web are part of the SW (they occur in some triples) even though they don’t have any semantic markup as such. Also, note that the diagram is slightly misleading. On the SW, these three web pages are resources identified by their URIs. So, the content of the three web pages is not part of the SW. For the content of a web page (say http://marysmith.net) to be a part of the SW, the following triple needs to be added to the SW.

```http
<http://www.marysmith.net http://www.somewhere.com#hasContent "...">```
In the above triple, the object is the content of the webpage (literal). The SW is expected to have many such triples with literals. The literal cannot be the subject of a triple therefore the literals are all terminal nodes of the SW. Therefore, the Semantic Web is not all structured data. It has islands of unstructured data residing in structured data. The content of this unstructured data (terminal web pages) is, once again, inaccessible to the machines. Retrieval of these web pages can be performed using SPARQL which allows regular expression matching.

Figure 4: The Hypertext web and the Semantic Web
For example, to search for course pages that talk about databases, the user or agent can compose the following query.

```
PREFIX univ: <http://www.univ.com/>  
Select ?g  
Where  
{  
  ?x rdf:type univ:course .  
  ?x univ:coursePage ?g .  
  FILTER regex(?g, "database", "i")  
}
```

In the above query, “i” is for case insensitive matching. This is classic Data Retrieval. The FILTER statement in the above query is akin to LIKE statement in SQL. This kind of text matching completely ignores the semantics of the content of the document and suffers from all the well documented problems that IR has been trying to address for the past 30 or so years.

But on the other hand, if the content of the documents is described using semantic descriptions (triples), then we can retrieve documents based upon the semantics of the content. Let us suppose that the triples of the following kind (Figure 5) exist in the database.

![Graph](http://www.cs.wright.edu-CS421/tables.jpg)

**Figure 5: Explicitly specifying the semantics of the document content**

Now, the user can explicitly query for pages that talk about databases as follows (assuming isAbout is part of the university domain).

```
PREFIX univ: <http://www.univ.com/>  
PREFIX cs: <http://www.csBranches.org/>  
Select ?g
```
Now, the question arises as to how the `isAbout` triples are added to the system. Obviously, they can either be added manually or they can be added by programs that extract content from the webpage and “understand” the content and automatically add the triples to the system. Following the second option basically means that we are back to square one trying to write programs that “understand” natural language. This problem of automatic information extraction (and triple generation) from natural language documents is not an easy one. It can perhaps be done on a small scale in selected domains but is not feasible in the general case especially on the highly heterogeneous WWW. Our best bet then would be to allow users to semi-automatically annotate documents (with meta-data that is indicative of the document semantics) that they are authoring. But then the question arises as to whether the new retrieval systems outperform systems like Google. Moreover, there will always be legacy documents without semantic annotations.

From the above examples, it can be seen that, like SQL, SPARQL requires its users to be intimately familiar with the schema information. It is reasonable for SQL to expect its users to have complete knowledge of the structure of the data (schema information) because RDBMS’ are typically used in corporate environment by highly skilled users who are intimately familiar with the structure of the not-so-heterogeneous data. In contrast, the SW is a vast heterogeneous database of information. Given the heterogeneity of the data, it is unreasonable to expect users to have intimate knowledge of the structure of the data. For instance, SPARQL expects users to know the exact URIs
of resources in order to compose queries. In a repository holding millions of URIs, this is an unreasonable expectation.

### 4.2.2 Limitations

As mentioned above, one of the major limitations of retrieval from SW is that SPARQL requires the user to be intimately familiar with the schema information. Another point is that it is not really clear how document retrieval can benefit from the Semantic Web technologies even though it was initially touted to enable true “Semantic Search”. But perhaps the most important concern is that of adding semantics to a document. Apparently, the only way of adding semantics to a document at present seems to be to involve its URL in a triple. In addition to all these, regular expression matching based retrieval of semi-structured RDF is simply not good enough and is a step backwards.

### 4.3 Objectives

So currently, documents are retrieved from the HW using IR techniques and data is retrieved from SW using DR techniques. But as seen from the previous sections, there are limitations to both these retrieval paradigms. We believe that these limitations can be addressed as follows:

1. Conceptually combine the two webs by clearly defining their interconnections.
2. View hyperlinks as semantic annotations.
3. Devise hybrid techniques to retrieve data and documents from this combined web.

This will result in the following advantages:
1. Document retrieval will improve, in terms of *precision*, because of the semantic annotations.

2. Document retrieval will improve, in terms of *recall* when synonymy (aliases and co-references) is involved.

3. Document retrieval will improve, in terms of *recall*, when the user performs broadening searches.

4. RDF documents (such as ontologies) and hybrid documents such as RDFa documents can be searched for and retrieved.

5. Anchor text enables keyword-based exploration of the RDF graph which can even lead to better SPARQL queries.

6. RDF graph can be queried even when schema information or exact URIs are not known.

7. Keyword-based search for semi-structured RDF data possible.

8. Enables “layman” querying of RDF in the future.

### 4.4 Hyperlinks as Semantic Markup

The SW is physically enclosed in web pages on the HW (as the RDF data is contained in files located on the Web). HTML markup tells the browser how to display a document. In contrast, semantic markup of content promotes its machine comprehension. Consider the following fragment from a document located at [http://www.one.com/A.html](http://www.one.com/A.html) that basically says that *B.html* is authored by John Smith.

```xml
<rdf:RDF>
  <rdf:Desc. rdf:about="http://www.two.com/B.html">
    <mydomain:author> John Smith </mydomain:author>
  </rdf:Desc.>
</rdf:RDF>
```
The physical location of this fragment (that is, the file in which it resides) is irrelevant to the resource that it is describing. So, “description” (or metadata) is a better term to describe this fragment than “markup”. There are systems that perform Semantic Web Document (SWD) retrieval on the Web viewing a document as a bag of URIs [D+05b]. This is akin to retrieving databases (as opposed to data) from the web based upon their contents. This approach makes sense for searching for ontologies and SW data (“retrieve documents that contain the URI XXX”), but is not appropriate for document retrieval because the location of the semantic description has nothing to do with the document that it is describing. What is needed for this bag of URIs model to be effective for document retrieval is markup technology that physically ties in the semantic description of a document with the document being described.

Keeping the above discussion in mind, we propose to treat hyperlinks as semantic markup. A hyperlink from a document to a node on the SW links the document to the node and at the same time annotates the document with the URI of the node. On the UW, it is likely that there will be hyperlinks from HTML documents to resources that are part of the SW (that is, participate in triples). We propose that this valuable information be utilized to enhance document retrieval from the UW. For example, if a document contains a hyperlink to mailto:bsmith@wright.edu, and if there is a triple in the database that tells us that <mailto:bsmith@wright.edu rdf:type univ:prof> then this information can be used to enhance document retrieval. Specifically, a search for an instance of a univ:prof can uncover the document containing mailto:bsmith@wright.edu. Effectively, ISA relationship encoded in the SW can be used to broaden the search results. Thus, a
hyperlink connecting an HTML page to the Semantic Web can be valuable from IR perspective.

Consider another example. On the web, it is not uncommon to see a document with hyperlinks from terms in the document to standard web pages (such as dictionary.com, Wikipedia, etc) that describe those terms.

“..The `<a href="http://dictionary.com/search?q=jaguar">Jaguar</a>`

God of the Underworld ….”

Here the hyperlink is from the term Jaguar to a webpage in an online dictionary [Jpage] that describes/defines the term. The dictionary webpage can be said to *annotate* the term Jaguar. Similarly, on the UW, the author of a webpage can provide a hyperlink to the appropriate URI to annotate a term as illustrated below.

“..The `<a href = "http://www.animalOnto.com/Jaguar">Jaguar</a>`

God of Underworld….“

This annotation is meant for machine agents rather than humans. This is a simple and elegant way of annotating a web page with SW data that can improve retrieval using the bag of URIs model. But it interferes with the human web navigation. To enable both human and machine consumption, we can use the combination.

```
```

Here the empty hyperlink (rendered invisible by the browsers) next to Jaguar captures the sense of the term Jaguar. However, this approach is not viable for legacy documents because it requires physical modification of the documents (similarly to what is enabled
by RDFa [AB06]). But, consider the following proposal of annotating the dictionary page defining *Jaguar*.

```
jaguar  <a href = "http://www.animalOnto.com/Jaguar"> </a>
```

A large feline mammal (*Panthera onca*) of Central and South America …

The empty hyperlink annotation with [http://www.animalOnto.com/Jaguar](http://www.animalOnto.com/Jaguar) can explicitly state the animal sense and disambiguate it from the potential car or football team sense. Now, pages that hyperlink to this dictionary page can be inferred to be relevant to *Jaguar* the animal context and [http://www.animalOnto.com/Jaguar](http://www.animalOnto.com/Jaguar) can be considered to annotate those pages. In summary, by adding annotations to the pages in a *single* web site (for example, dictionary.com), we can annotate a host of legacy documents which link to the pages on the web site. This is an improvement over the previous approach but requires modification of the dictionary.com pages. We can further achieve scalability for extant legacy documents simply by adding the following triple to the IR system’s database.

```
```

This information can be used to conclude that the (unmodified) web pages linking to [http://dictionary.com/search?q=jaguar](http://dictionary.com/search?q=jaguar) (which is also unmodified) are talking about Jaguar, the animal. This idea can be extended to create ontology websites where each web page corresponds to an entity in the ontology. A user can annotate a document simply by adding a hyperlink to one of the pages in the web site. A web page can be considered to have semantic annotation simply because it has a hyperlink to a Semantic Web data node or because it is linking to another web page that has explicit semantic annotations. Therefore, the existing hyperlink structure can be harnessed and used in conjunction with semantic descriptions to enhance document retrieval. The UW provides a framework where this is possible (due to the *linksTo* relationship).
5 The Unified Web Model

The Unified Web model aims to integrate the SW and the HW into a single unified whole by encoding the two webs and the connections between them. The UW model is a graph of nodes and edges \((N, E)\).

5.1 Node

Node is an abstract entity that is uniquely identified by its URI. A Node may or may not have a document associated with it. But there is at least one node (and hence one URI) associated with a document. A document is a concrete container of information. A node can be seen as an abstract container that “contains” the following categories of information.

The “Home URI” section contains the textual representation of the URI of the node. Additionally, it contains a bag of words and phrases called “URI Index Words” or UIW constituted from various sources. For example, the words can be substrings of the URI, or come from the object literal of a triple whose subject is the URI and predicate is, say, \(rdfs:label\). They can come from the anchor text of the URI in some document. For example, the hyperlink \(<a \ href= “mailto:bsmith@wright.edu”>William</a>\) can contribute William to the UIW of the node whose URI is mailto:bsmith@wright.edu. The “Document” section contains the textual representation of the document associated with the node (if any). The “Parameters” section contains information about the document such as filename, date of creation, etc., which is usually not a part of the document itself and should be obtained from the server serving the document. The “External Text”
section contains fragments of text from other documents (for example anchor text) whose
nodes participate in a linksTo relationship (to be discussed below) with the current node.
This section may have words in common with the UIW. The “Outgoing Links” section
contains URIs of nodes to which the current node has an outgoing linksTo link. The
“Triples” section contains the textual representation of the triples asserted by the node.

The above “container” of information associated with a node is the information
that the retrieval system should keep track of. The “Home URI”, the “Document” and the
“External text” sections can be represented as bags of words. The “Outgoing links” and
the “Triples” sections can be represented as a bag of URIs and a bag of triples
respectively. All of these serve to “annotate” the node and can be used to index the node
for retrieval.

There are two categories of nodes: (i) Natural nodes (NN) and (ii) System defined
nodes (SN). The natural nodes can be further classified as plain (or non-document) nodes
(PN) and document nodes (DN) based on whether or not a node has an associated
document. The system defined nodes can be further classified as literal nodes (LN), triple
nodes (TN) and blank nodes (BN). The system creates a URI and assigns it to each blank
node, triple and literal that it encounters on the Web.

5.2 Edges (Relationships)
There are two categories of edges: (i) User defined edges (UE) and (ii) System defined
edges (SE). The user defined edges come from the triples in the (Semantic Web)
documents while the system defined edges are defined to make explicit the
interconnections between the HW and the SW. The system defined edges are the
following. The *asserts* edge exists from a node (document) to each of the RDF statements found in the associated document. The RDF statement itself has a subject, a property and an object. There is no restriction as to how a triple is obtained from the document. The *hasDocument* edge exists from a node to a literal. The literal is the string representation of the document associated with the node. A *hyperlinksTo* edge exists from a node A to another node B if there is a hyperlink from the document of node A to the document of node B. The *linksTo* edge exists from node A to node B if a *hyperlinksTo* relationship exists from node A to node B, or node B occurs in any of the triples asserted by node A (see Figure 6).

![Figure 6. Relationships](image)

More formally, they can be specified as functions/relations in terms of their signatures (domains and ranges), and include:

\[
\text{hyperlinksTo} \subseteq DN \times NN
\]
\[ \text{linksTo} \subseteq DN \times NN \\]
\[ \text{asserts} \subseteq DN \times TN \\]
\[ \text{hasDocument}: DN \rightarrow LN \]

These relations are not independent and cannot be assigned arbitrarily. They must satisfy at least the following constraints.
\[ \forall n \in DN, \forall m \in NN: \text{ if } [n, \text{hyperlinksTo}, m] \in SE \text{ then } [n, \text{linksTo}, m] \in SE. \]

The Unified Web model is not a simple super-imposition of the SW graph over the hypertext graph. The Semantic Web can be thought of as a global RDF graph constructed by gathering all possible RDF triples from documents that reside on the Hypertext Web. The UW reifies each of the SW triples by explicitly encoding the \textit{asserts} relationship between a document and the triple that is extracted from it. The UW can be visualized as a meta – Semantic Web which in itself can be an RDF graph (one that subsumes all RDF graphs found on the web). In addition, this RDF graph also encodes the Hypertext Web (HTML documents and hypertext links between them). The aim is to encode the HW (\textit{hyperlinksTo} and \textit{hasDocument}) and the SW and the connections between the two (such as \textit{asserts} which is not explicitly defined by the user but is rather constructed by the system) while allowing easy mapping of data retrieval queries meant for the “conventional” SW to those for the UW. The \textit{linksTo} tries to define a generic “connection” between two nodes. It seeks to establish a definite connection between a document node and a SW data node (which, of course, can be a document node as well) and to deliberately blur the distinction between such a connection and a hypertext connection. The \textit{linksTo} edge is for the Unified Web what the hyperlink is for the HW and the property-link is for the SW. In our implementation, we use \textit{linksTo} to view a
document as being annotated by the URIs that it \textit{linksTo} and use this information while retrieving documents.

\textbf{5.3 UW Model in RDF}

The UW model can be specified using RDF and the system can be implemented as an RDF database. The resource (node) in the model can be anything. A system implementing the model can treat a node as an equivalent of \textit{rdfs:Resource} [BG04]. The document nodes can simply be treated as data nodes. The document nodes participate in \textit{hasDocument} relationships – but as far as the system is concerned, they are just data nodes. The relationships can only be one of the following: \textit{asserts, linksTo, hyperlinksTo} and \textit{hasDocument}. The following RDF/XML describes the model.

```xml
<rdf:RDF
  xmlns="http://www.system.org/web#"
  xml:base="http://www.system.org/web">
  <owl:Class rdf:ID='Node'>
  </owl:Class>
  <rdf:Property rdf:ID='asserts'>
    <rdfs:domain rdf:resource='http://www.system.org/web#Node'/>
    <rdfs:range rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement'/>
  </rdf:Property>
  <rdf:Property rdf:ID='hasDocument'>
    <rdfs:domain rdf:resource='http://www.system.org/web#Node'/>
    <rdfs:range rdf:resource='http://www.w3.org/2000/01/rdf-schema#Literal'/>
  </rdf:Property>
  <rdf:Property rdf:ID='HyperlinksTo'>
    <rdfs:domain rdf:resource='http://www.system.org/web#Node'/>
    <rdfs:range rdf:resource='http://www.w3.org/2000/01/rdf-schema#Literal'/>
  </rdf:Property>
  <rdf:Property rdf:ID='linksTo'>
    <rdfs:domain rdf:resource='http://www.system.org/web#Node'/>
    <rdfs:range rdf:resource='http://www.system.org/web#Node'/>
  </rdf:Property>
</rdf:RDF>
```

The namespace of this document is the namespace of the system itself (in our case the fictional \url{http://www.system.org/web}). The complete document can be found in Appendix A. The relationships \textit{rdf:Subject, rdf:Predicate, rdf:Object}, naturally exist between an
RDF statement and its components. The asserts, linksTo, hyperlinksTo and hasDocument are called system relationships and the statements involving these relationships are implicitly asserted by the system and are called system triples. The system asserts these statements by exploring the repository and parsing the documents in the repository. These statements, along with the statements involving rdf:Subject, rdf:Predicate, rdf:Object form the UW. Therefore, the UW can be implemented as a database of triples.

The UW meets the definition of the SW. It is a web of nodes linked by properties whose semantics are well defined. Agents can reason with the data on UW. However, it is one layer of abstraction above the “conventional” view of the Semantic Web. UW is reified SW. But from the agent perspective, the (conventional) SW is just one query away from the UW. The following query returns all the statements that form the SW.

```
SELECT ?x
WHERE {?y asserts ?x}
```

So, the UW encodes the SW. But to avoid confusion, we will make the distinction between the UW and the SW. Every RDF statement in the system has to be asserted by some entity. The statements asserted by the resources (as opposed to the system) are simply called user triples and will be simply referred to as triples for the sake of brevity. These triples form what is conventionally known as Semantic Web. As will be explained shortly, the system allows the users to retain this perspective of the SW and compose queries for this SW.

5.3.1 Retrieving from UW using DR techniques

The model allows the users to retain the logical (conventional) conceptualization of the Semantic Web and pose queries to it using the SW query languages like SPARQL. The
SPARQL queries for the SW can easily be “transformed” to SPARQL queries for the UW.

For example, the following “conventional” query

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1>
}
```

Can easily be transformed into:

```
SELECT ?title
WHERE
{
}
```

Therefore, retrieving data from the UW using SPARQL queries is straightforward. But note that since every triple is encoded as an `rdf:Statement`, there is additional computation involved in processing the query. But a system implementing the model can be optimized in obvious ways.

As explained before, web documents appear as literals on the SW. In the same way, web documents appear as literals on the UW. We can use SPARQL queries to retrieve documents from the UW using DR methods. For example, the following SPARQL query searches for documents containing the word “red”. The parameter “i” indicates that the match is not case-sensitive.

```
Select ?x
Where
{
  ?x hasDocument ?g .
  FILTER regex(?g, "red", "i")
}
```
The user can also utilize additional information to search for documents on the UW. For example, the following query searches for documents authored by John and have the word “jaguar” in them.

Select ?x
Where
{
  ?x http://purl.org/dc/elements/1.1/creator mailTo:john@abc.com .
  FILTER regex(?x, "jaguar", "i")
}

Which would then map to the system query:

Select ?x
Where
{
  ?y rdf:subject ?x .
  ?y rdf:object mailTo:john@abc.com .
  ?x hasDocument ?g .
  FILTER regex(?g, "jaguar", "i")
}

But, as explained earlier, this type of regular expression based document search is nothing but text matching and hence DR. Performing such simple regular expression matches to retrieve string literals (documents) from a repository as vast and heterogeneous as UW (or SW for that matter) is simply not good enough. At best, this corresponds to the Boolean retrieval model of IR.

Hence the model allows the users to pretty much do whatever they have been doing on the SW. The users can perform data and document retrieval using the query languages that they have been using before. All that is needed is a layer that maps SPARQL queries for the SW to SPARQL queries for the UW. The mapping is expected to be simple and straight forward. A rigorous, exhaustive, formal mapping procedure from SW SPARQL queries to the UW SPARQL queries is outside the scope of our work.
6 Hybrid Query Language

As mentioned before, our goal is to conceptually unify the two webs and to retrieve data and documents from this unified web using hybrid techniques. The previous chapter discussed the Unified Web model that combines the two webs. In this chapter we describe the Hybrid Query Language (HQL) that combines the Information and Data retrieval paradigms to enable hybrid retrieval of the data and documents from the Unified Web. One of the fundamental features of HQL is the concept of wordset. The wordset enables the wordset pair which is a unique way of combining keyword-based search with reasoning.

6.1 Wordsets

The wordsets allow users to search for nodes (URIs) based upon the words and phrases in their URI Index Words (UIW). A “word set” is a set of words and phrases (multiple words enclosed in quotes) enclosed in angular brackets as shown in the example below.

Example: <Microsoft bill research>

Given a word set, the system retrieves all the nodes in the UW such that all of the words in the word set appear in the node’s UIW. For example, let the Home URI of a node be mailto:bsmith@microsoft.com. Let this node be referenced from another HTML document as follows:

    <a href=mailto:bsmith@microsoft.com> Research Scientist </a>

Also, let the following triple be asserted by some node.

    <mailto:bsmith@microsoft.com rdfs:label “William Smith”>
Then the UIW of the node will (perhaps) be: \{“bsmith” “microsoft” “Research Scientist” “Research” “Scientist” “William Smith” “William” “Smith”\}. This node will be retrieved by the query \(<Smith Research>\), but not by \(<Smith Research Bill>\). Thus multiple words inside angular brackets have implicit conjunction.

6.2 Wordset Pairs

The user can provide the system with disambiguation information using the novel wordset pair queries. A wordset pair consists of two wordsets separated by the scope resolution operator as shown below:

Example: \(<animal>::<jaguar>\)

The first of the pair refers to the class/superclass and the second of the pair to the instance/subclass. So the wordset pair \(<person>::<john>\) would match a node whose UIW contains “john” and which is a direct or indirect instance of a URI whose UIW contains “person”. We now give a formal description of the query language.

6.3 Formal Description of HQL

The Hybrid Query Language enables convenient navigation and extraction of information from the UW. It enables formulation of precise queries involving URIs, and “approximate” word-based queries that capture context (e.g., wordset, wordset-pairs queries) and/or content (e.g., keyword queries). In other words, it enables access to both HW documents and SW data, incorporating indexing information from the neighboring nodes. Specifically, the wordset queries can use anchor text in the HW to retrieve SW
nodes, and wordset pair queries can express disambiguation information using the ISA edges encoded in the SW for semantic search of HW documents.

Before we go into the details of the query language, let us first define some more utility functions/relations in addition to the four in the previous section:

\[
\begin{align*}
\text{homeURI}: & \ N \rightarrow \ Set(URI) \\
\text{indexWords}: & \ NN \rightarrow \ PowerSet(STRINGS) \\
\text{hasTriples}: & \ DN \rightarrow \ PowerSet(NN \times NN \times NN) \\
\text{hasLiteral}: & \ LN \rightarrow \ STRINGS
\end{align*}
\]

URI denotes a string that must satisfy the URI syntax requirement (RFC 3986), while STRINGS denotes a set of words, phrases, and other fragments. PowerSet operator yields a set of all subsets. The members of NN x NN x NN are referred to as the triples (such as those found in RDF documents).

homeURI maps a node to its URI. The URI is what we use to refer to a node explicitly. indexWords maps a node to a set of strings that can serve as an index to it. These can be composed from the URI and the anchor text from the neighboring nodes among other things. hasDocument maps a document node to the associated document text string. hyperlinksTo relates a document node to a node that appears in a hyperlink in the corresponding document. linksTo relates a document node to a node that appears in the corresponding document. This can be in the form of a hyperlink or embedded in a triple. hasTriples maps a document node to the set of 3-tuples of nodes that appear in the corresponding document. asserts relates a document node to a triple node that reifies the triple that appears in the corresponding document. hasLiteral maps a literal node to the string it is associated with. It is possible to have multiple literal nodes associated with the same string. Note that a specific instantiation of the framework can be obtained by
defining how these functions/relations (such as externalText, IndexWords, etc) are obtained from the node’s neighborhood.

These functions must satisfy at least the following constraints.

\[ \forall n \in NN, \forall [n_1, n_2, n_3] \in NN \times NN \times NN: \]

\[ [n_1, n_2, n_3] \in \text{hasTriples}(n) \quad \text{only if} \]

\[ [n, \text{linksTo}, n_1] \in SE \land [n, \text{linksTo}, n_2] \in SE \land [n, \text{linksTo}, n_3] \in SE \]

\[ \forall n \in N, [n_1, n_2, n_3] \in NN \times NN \times NN: \]

\[ [n_1, n_2, n_3] \in \text{hasTriples}(n) \quad \text{if and only if} \]

\[ \exists \, \text{tn} \in TN: [n, \text{asserts}, \text{tn}] \in SE \land [\text{tn}, \text{rdf:subject}, n_1] \in SE \]

\[ \land [\text{tn}, \text{rdf:predicate}, n_2] \in SE \land [\text{tn}, \text{rdf:object}, n_3] \in SE \]

For convenience, we abuse the language and say that \( n_1, n_2, \) and \( n_3 \) appear in \( \text{tn} \) in the context of the reification constraint. We also abbreviate \([n, \text{property}, m] \in SE\) as \([n, \text{property}, m]\).

In what follows, we motivate and specify the abstract syntax of the queries using a context-free grammar, and the semantics of the queries in terms of the Unified Web model, in sufficient detail to enable prototyping. Our presentation focuses on queries that yield a set of nodes. The “domain information bearing” strings such as the document text, literal, etc. can be easily obtained from a URI by calling corresponding system functions such as \( \text{hasDocument}, \text{hasLiteral}, \) etc. and from triples using \( \text{rdf:subject}, \text{rdf:predicate}, \) and \( \text{rdf:object}, \) etc.
TopLevelQuery ::= Nodes-ref | Triples-ref | ...

QUERY: Nodes-ref ::= u, where u ∈ Set(URI).

ANSWER: Result(u) = \{ n ∈ N | HomeURI(n) = u \}

SEMANTICS: The URI-query returns the set containing the unique node whose
HomeURI matches the given URI. Otherwise, it returns an error.

EXAMPLE: http://www.aifb.uni-karlsruhe.de/Personen/viewPersonenglish?id_db=20

QUERY: Nodes-ref ::= ss, where ss ∈ PowerSet(STRINGS).

ANSWER: Result(ss) = \{ n in N | ss ⊆ IndexWords(n) \}

SEMANTICS: The wordset query, ss, usually written as a set of strings delimited using
angular brackets, returns the set of nodes whose IndexWords contain ss.

EXAMPLE: <peter haase>

QUERY: Nodes-ref ::= pp::ss, where pp, ss ∈ PowerSet(STRINGS).

ANSWER: Result(pp::ss) =

\{ n ∈ N | ss ⊆ IndexWords(n) ∧ ∃m : n ISA m ∧ pp ⊆ IndexWords(m) \}

SEMANTICS: The wordset-pair query, pp::ss, usually written as two wordsets delimited
using colon, returns the set of nodes such that each node has IndexWords that contains ss
and has an ISA ancestor whose IndexWords contains pp.

EXAMPLE: <student>::<peter>
QUERY: \( \text{Triples-ref} ::= u, \quad \text{where } u \in \text{Set(URI)}. \)

ANSWER: \( \text{Result}(u) = \{ n \in TN \mid \text{HomeURI}(n) = u \} \)

SEMANTICS: The *triple node URI-query* returns the set containing the unique node whose *HomeURI* matches the given triple node URI. Otherwise, it returns an error. The triple nodes are system generated.

EXAMPLE:

http://www.aifb.uni-karlsruhe.de/Persoenen/viewPersonFOAF/foaf_80.rdf#tri52

QUERY: \( \text{Triples-ref} ::= \text{Single-Var-Triples-ref} \)

\[
\text{Single-Var-Triples-ref} ::= [\?var \text{ Nodes-ref} \text{ Nodes-ref}]
\]

\[
\text{Single-Var-Triples-ref} ::= [\text{Nodes-ref} \ ?var \text{ Nodes-ref}]
\]

\[
\text{Single-Var-Triples-ref} ::= [\text{Nodes-ref} \text{ Nodes-ref} \ ?var]
\]

where \(?var\) is a variable.

ANSWER: \( \text{Result}([\?var \text{ Nodes-ref1} \text{ Nodes-ref2}]) = \)

\[
\{ t \in TN \mid n_1 \in \text{Result}(	ext{Nodes-ref1}) \land n_2 \in \text{Result}(	ext{Nodes-ref2}) \\
\land \exists m \in N : [m, \text{asserts}, t] \land [t, \text{rdf:predicate}, n_1] \land [t, \text{rdf:object}, n_2] \}
\]

Similarly, for the other two cases.

SEMANTICS: The *part triple query* \([?var \text{ Nodes-ref1} \text{ Nodes-ref2}]\) returns the set of (system generated) triple nodes that contain a node related by a binary predicate denoted by *Nodes-ref1* to some node denoted by *Nodes-ref2*. Similarly, for the other two cases.

Note that this query characterizes a node using its neighborhood.

EXAMPLE: \([<silver> \ <\text{atomic weight}> \ ?x]\)
QUERY: \( \text{Triples-ref ::= [Nodes-ref, Nodes-ref, Nodes-ref]} \)

ANSWER: \( \text{Result([Nodes-ref1, Nodes-ref2, Nodes-ref3]) =} \)
\[
\{ t \in TN \mid n1 \in \text{Result(Nodes-ref1)} \land n2 \in \text{Result(Nodes-ref2)} \land n3 \in \text{Result(Nodes-ref3)} \land \exists m \in N : [m, asserts, t] \land [t, rdf:subject, n1] \land [t, rdf:predicate, n2] \land [t, rdf:object, n3] \}
\]

SEMANTICS: The full triple query \([\text{Nodes-Ref, Nodes-Ref, Nodes-ref}]\) returns the set of (system generated) triple nodes matching the node references.

EXAMPLE:

[ http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2062instance <name> <peter> ]

QUERY: \( \text{Triples-ref ::= Double-Var-Triples-ref} \)

\( \text{Double-Var-Triples-ref ::= [?var, ?var, Nodes-ref]} \)

\( \text{Double-Var-Triples-ref ::= [?var, Nodes-ref, ?var]} \)

\( \text{Double-Var-Triples-ref ::= [Nodes-ref, ?var, ?var]} \)

where \(?\text{var}\) is a variable.

ANSWER: \( \text{Result([?var, ?var, Nodes-ref]) =} \)
\[
\{ t \in TN \mid m \in \text{Result(Nodes-ref)} \land [t, rdf:object, m] \land \exists n \in N : [n, asserts, t] \}
\]

Similarly, for the other two cases.

SEMANTICS: The part triple to triples query \([?\text{var} \ ?\text{var} \ Nodes-ref]\) returns the set of (system generated) triple nodes that contain some node denoted by Nodes-ref. Similarly,
for the other two cases. Note that this query characterizes the node neighborhood. Each variable occurrence is independent of the other occurrences.

EXAMPLE: $[?x <title> ?x]$ 

QUERY: $\text{Nodes-ref} ::= \text{Nodes-ref AND Nodes-ref}$

$\text{Nodes-ref} ::= \text{Nodes-ref OR Nodes-ref}$

$\text{Triples-ref} ::= \text{Triples-ref AND Triples-ref}$

$\text{Triples-ref} ::= \text{Triples-ref OR Triples-ref}$

SEMANTICS: “OR” and “AND” are interpreted as set-union and set-intersection respectively. Each variable occurrence is independent of the other occurrences.

6.3.1 Queries for Exploring the System-generated Neighborhood of a Node

QUERY: $\text{Nodes-ref} ::= \text{getAllTriples}(\text{Nodes-ref})$

ANSWER: $\text{Result( getAllTriples}(\text{Nodes-ref})) =$

$$\{ t \in TN \mid n \in \text{Result(Nodes-ref)} \land n \text{ appears in } t$$

$$\land \exists m \in N: [m, asserts, t] \}$$

SEMANTICS: This query retrieves the (system generated) triple nodes in which the queried node URI appears.

EXAMPLE:

gGetAllTriples($http://www.daml.org/2003/01/periodictable/PeriodicTable#group_11$)

QUERY: $\text{Nodes-ref} ::= \text{getLinkingNodes}(\text{Nodes-ref})$
ANSWER: \[ \text{Result}(\ \text{getLinkingNodes}(\text{Nodes-ref})) = \]

\[ \text{Result}(\text{Nodes-ref}) \cup \]

\[ \{ \ m \in N \mid \exists n \in \text{Result}(\text{Nodes-ref}) : [m, \text{linksTo}, n] \} \]

SEMANTICS: This query retrieves the nodes corresponding to \text{Nodes-ref} and the document nodes containing references to the nodes corresponding to \text{Nodes-ref}. Effectively, nodes and their neighborhoods are retrieved.

EXAMPLE: \text{getLinkingNodes(http://www.aifb.ukarlsruhe.de/Personen/viewPerson?id\_db=2023 )}

QUERY: \text{Nodes-ref ::= getAssertingNodes(Triples-ref)}

ANSWER: \[ \text{Result}(\ \text{getAssertingNodes}(\text{Triples-ref})) = \]

\[ \{ \ m \in DN \mid \exists t \in \text{Results}(\text{Triples-ref}) : [m, \text{asserts}, t] \} \]

SEMANTICS: This query retrieves document nodes containing the triples.

EXAMPLE: \text{getAssertingNodes([<peter haase> <publication> ?x])}

QUERY: \text{Nodes-ref ::= getDocsByKeywords(ss), where ss \in PowerSet(STRINGS).}

ANSWER: \[ \text{Result}(\ \text{getDocsByKeywords}(\text{kws})) = \]

\[ \{ \ m \in DN \mid \text{hasDocument}(m) = dt \land \text{match}(kws, dt) \} \]

SEMANTICS: This query is analogous to the traditional keyword query that takes a set of keywords and retrieves document nodes that match the keywords. \text{match} embodies the criteria for determining when a document text is “relevant” to a keyword. It can be as simple as requiring verbatim occurrence, to as complex as requiring stemming, synonym
generation, spelling correction, etc. *match* may be *compositional*, that is, \( \text{match} \) (kws, dt) = \( \forall w \in \text{kws}: \text{match}(w, dt) \), but it is not required.

**QUERY:** \( \text{Nodes-ref} ::= \text{getLiteralsByKeywords} \text{(ss)}, \) where \( \text{ss} \in \text{PowerSet(STRINGS)}. \)

**ANSWER:** \( \text{Result( getLiteralsByKeywords(kws) )} = \) \( \{ m \in \text{LN} \mid \text{hasLiteral}(m) = \text{dt} \land \text{match(kws, dt)} \} \)

**SEMANTICS:** This is analogous to the above query customized for literal nodes.

**EXAMPLE:** \( \text{getLiteralsByKeywords(semantic grid)} \)

### 6.3.2 Further Queries for Retrieving Documents

**QUERY:** \( \text{getDocsByContent: PowerSet(STRINGS) -> PowerSet(DN)} \)

**ABBREVIATION FOR:**
\[
\text{getDocsByContent(kws)} = \text{getLinkingNodes(getDocsByKeywords(kws))}
\]
where \( \text{kws} \in \text{PowerSet(STRINGS)}. \)

**SEMANTICS:** This query retrieves the document nodes with content matching keywords in kws and the neighboring document nodes that reference such nodes. Intuitively, we want to pursue both the “authorities” and the “hubs” [K98], assisting both navigational searches and research searches [GMM03].

**QUERY:** \( \text{getDocsByIndexOrContent: PowerSet(STRINGS) -> PowerSet(DN)} \)

**ABBREVIATION FOR:** \( \text{getDocsByIndexOrContent (kws) } = \) \( \text{getDocsByKeywords(kws)} \lor \bigvee_{\text{kws} \in \text{kws}} \text{getLinkingNodes(kw)} \)
where kws ∈ PowerSet(STRINGS).

SEMANTICS: This query retrieves the document nodes with content matching the keywords kws or in the neighborhood of nodes indexed by kws. Implicitly, the former captures syntactic retrieval and the latter enables semantic retrieval.

EXAMPLE: getDocsByIndexOrContent(semantic web)

QUERY: getDocsByIndexAndContent:

\[
\text{Nodes-ref} \times \text{PowerSet}(\text{STRINGS}) \rightarrow \text{PowerSet}(\text{DN})
\]

ABBREVIATION FOR: getDocsByIndexAndContent (nr, kws) =

\[
\text{getLinkingNodes(Result(nr))} \land \text{getDocsByKeywords(kws)}
\]

where nr ∈ Nodes-ref, kws ∈ PowerSet(STRINGS).

SEMANTICS: This query retrieves the document nodes with content matching the keywords kws and in the neighborhood of nodes corresponding to nr. Implicitly, if nr is a URI of a document node containing the keywords kws, then the result will contain this document node. If nr is a URI and this URI and the keywords kws are contained in a document, then the result will contain the latter document node. Similarly, for nodes in Result(nr) when nr contains wordset and wordset-pairs.

EXAMPLE: getDocsByIndexAndContent(<car>::<jaguar>)

EXAMPLE: getDocsByIndexAndContent(<car>::<jaguar> engines)

QUERY: getDocsByTriplesAndContent:

\[
\text{Triples-ref} \times \text{PowerSet}(\text{STRINGS}) \rightarrow \text{PowerSet}(\text{DN})
\]
ABBREVIATION FOR: \[ \text{getDocsByTriplesAndContent}(\text{tr}, \text{kws}) = \]

\[ \text{getAssertingNodes}(\text{tr}) \land \text{getDocsByKeywords}(\text{kws}) \]

where \( \text{tr} \in \text{Triples-ref} \), \( \text{kws} \in \text{PowerSet(STRINGS)} \).

SEMANTICS: This query retrieves (Semantic Web) document nodes that match the keywords and contain the referenced triples.

QUERY:

\[ \text{Single-Var-Triples-list ::= Single-Var-Triples-ref} \]

\[ \text{Single-Var-Triples-list ::= Single-Var-Triples-ref} \]

\[ \text{Single-Var-Triples-list} \]

\[ \text{Nodes-ref ::= getBindings(Single-Var-Triples-list)} \]

QUERY1: \[ \text{Nodes-ref ::= getBindings([?var Nodes-ref Nodes-ref])} \]

ANSWER: \[ \text{Result(getBindings([?var Nodes-ref1 Nodes-ref2])} = \]

\[ \{ n \in N \mid n1 \in \text{Result(Nodes-ref1)} \land n2 \in \text{Result(Nodes-ref2)} \]

\[ \land \exists m \in N : [m, \text{asserts, t}] \land [t, \text{rdf:subject, n}] \]

\[ \land [t, \text{rdf:predicate, n1}] \land [t, \text{rdf:object, n2}] \}

Similarly, for the other two cases.

QUERY2: \[ \text{Nodes-ref ::= getBindings(Single-Var-Triples-ref} \]

\[ \text{Single-Var-Triples-list)} \]

ANSWER: \[ \text{Result(getBindings(Single-Var-Triples-ref Single-Var-Triples-list))} = \]

\[ \text{Result(getBindings(Single-Var-Triples-ref))} \land \]

\[ \text{Result(getBindings(Single-Var-Triples-list))} \]
SEMANTICS: This query retrieves the bindings for the variables that satisfy all the triple references with single variable. All the variable occurrences are considered identical, that is, they must all be assigned the same value throughout the getBindings-argument.

EXAMPLE: getBindings([<phdstudent>::<peter> <name> ?x])

EXAMPLE: getBindings( [?x <group> <group 1>] [?x <color> <white>])

QUERY: getDocsByBindingsAndContent:

\[
\text{getDocsByBindingsAndContent}: \quad \text{Single-Var-Triples-list} \times \text{PowerSet}(\text{STRINGS}) \rightarrow \text{PowerSet}(\text{DN})
\]

ABBREVIATION FOR: getDocsByBindingsAndContent(vtl, kws) =

\[
\text{getBindings (vtl)} \land \text{getDocsByKeywords(kws)}
\]

where \( vtl \in \text{Single-Var-Triples-list} \),

\( kws \in \text{PowerSet}(\text{STRINGS}) \).

SEMANTICS: This query retrieves document nodes that match the keywords and contain the matching triples.

EXAMPLE: getDocsByBindingsAndContent([<phdstudent>::<peter> <homepage> ?x]

"Semantic Grid")
7 Implementation and Evaluation

7.1 Implementation

We have implemented a system called SITAR (Semantic InformaTion Analysis and Retrieval system) based upon our model. SITAR is built upon Apache Lucene [Luc] and stores data and documents persistently in Lucene indexes. Currently the system can index text, RDF/OWL and HTML files in addition to RDF triples. It uses CyberNeko HTML parser [Cyb] to parse HTML documents and the Jena ARP [Jen] parser to parse RDF documents and extract triples from them. At present, the system does not support pdf, doc files etc. (we index their URIs but their content is not being analyzed).

![SITAR Architecture](image)

**Figure 7: SITAR Architecture**

The general architecture of the system is shown in Figure 7. The crawler collects RDF/HTML/text documents from the Web and stores the documents in a local cache.
The documents are then parsed by the appropriate parser and the output of the parser is analyzed and indexed by Lucene. Every URI that is encountered (in HTML or RDF files) is analyzed using several heuristics to build a set of index words for it. More importantly, if the URI occurs in a HTML document as a hyperlink, we use the anchor text to add to its index words set. A HTML document is indexed by the URIs that it linksTo (includes hyperlinksTo) as well as the words that are extracted from its body. (Recall that a HTML document linksTo an URI if the URI is present in the document.) In this sense, it can be seen as a bag of URIs and words. An RDF document is indexed by the URIs that it linksTo and by the triples (URIs) that it asserts. In this sense, an RDF document can be seen as a bag of URIs and triples. Note that a hybrid document such as an RDFa document would be indexed by all of the above. But we are not experimenting with RDFa documents at present. The triples extracted from an RDF document are also stored in Lucene indexes. The system stores and processes URIs using their CRCs.

7.2 Evaluation

SITAR indexes RDF, HTML and text documents. Each document is indexed using words, URIs and triples that are extracted from it. HQL provides a mechanism to retrieve the documents viewing each one of them as a bag of words, URIs and triples. So, if all the documents in a given dataset are plain text documents (i.e., without URIs and Triples), then SITAR is no different from a standard search engine (such as Lucene) when it comes to document retrieval. The HQL query getDocByKeywords is essentially a Lucene keyword search. But the idea behind SITAR is to exploit semantic information
when available to improve document search. In the Unified Web, the semantic information can be associated with a document in two ways:

1. The document (i.e., its URL) participates in a triple.

2. The document has an outgoing \textit{LinksTo} link with a node that participates in a triple. (Note that in some cases, the triple may be asserted by the document itself).

When semantic information is associated with the documents, the document retrieval can benefit as follows:

1. User can query and retrieve RDF, HTML as well as hybrid documents (RDFa) by posing queries such as retrieve Peter’s homepage or retrieve the document asserting triples about Peter.

2. The \textit{precision} should improve if the user is able to convey her information need to the system unambiguously.

3. Since a node is indexed by words obtained from its neighboring nodes (such as literals of triples in which it participates, anchor text, etc.) the \textit{recall} can improve especially when synonymy is involved.

4. Broadening searches such as obtain pages talking about boxers should result in improved \textit{recall}.

In order to experiment with and evaluate SITAR we needed a document set that had HTML pages as well as RDF annotations and descriptions. The AIFB SEAL [HS04] document set turned out to be a good match for our needs. The AIFB SEAL website has human-consumable XHTML documents (in English and German) along-side RDF/OWL documents. Every entity, such as person, research topic, project, publication, etc., has a URI and an RDF/OWL file associated with it. The URI also acts as the URL of the
RDF/OWL page. The RDF file contains information about the entity. For example, if the entity is a person, the RDF file contains the person’s name, designation, projects she is working on, publications of the person and so on. The entity typically has an English and a German XHTML webpage (we can call it homepage) associated with it. Essentially the homepages are renderings of data retrieved from a relational database (though this is irrelevant for our purposes). For example, the following are the different URIs associated with the entity (research project) called **AIFB SEAL**:

- URI/OWL page: [http://www.aifb.uni-karlsruhe.de/Projekte/viewProjektOWL/id58instance](http://www.aifb.uni-karlsruhe.de/Projekte/viewProjektOWL/id58instance)
- English homepage: [http://www.aifb.uni-karlsruhe.de/Projekte/viewProjektenglish?id_db=58](http://www.aifb.uni-karlsruhe.de/Projekte/viewProjektenglish?id_db=58)
- German homepage: [http://www.aifb.uni-karlsruhe.de/Projekte/viewProjekt?id_db=58](http://www.aifb.uni-karlsruhe.de/Projekte/viewProjekt?id_db=58)

We crawled the SEAL website looking only for English versions of web pages and RDF/OWL files using heuristics. We collected and indexed a total of 1455 (610 RDF files and 845 XHTML files). A total of 193520 triples were parsed and indexed though there is no guarantee that the same triple was not asserted by two different documents. Note that, all the index structures are persistently stored. We now present some qualitative and quantitative results to give an idea about the dataset and about the kind of queries that can be formulated using HQL.

### 7.2.1 Qualitative Analysis

Let us suppose that the user is searching for information about a person named Peter. The user can pose the wordset query `<peter>` which in effect returns all nodes (URIs) that have been indexed using the word *peter*. A total of 52 URIs were retrieved in response to the above query including OWL files (instance data of people) and HTML files. The user can convey to the system that she is looking specifically for Ph.D. students named *peter*
using the query `<phdstudent>::<peter>`. The following URIs were retrieved in response to this query:

- `http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2023instance`
- `http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2119instance`
- `http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2062instance`

These are URIs (of OWL files) representing individuals and containing information about them. In order to find out the names of these individuals, the user can use the query `getBindings([<phdstudent>::<peter> <name> ?x])`. This query returned 125 literal nodes gathered from different RDF files (apparently FOAF files). Note that the above queries are keyword-based, and hence easy to formulate, and enable transparent traversal of the SW. The system finds the bindings for the variable from triples such as those shown below:

```
uri: http://www.aifb.uni-karlsruhe.de/Personen/viewPersonFOAF/foaf_80.rdf#tri52
sub: http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2023instance
pred: http://xmlns.com/foaf/0.1/name
obj (Literal): Peter Haase

uri: http://www.aifb.uni-karlsruhe.de/Personen/viewPersonFOAF/foaf_2127.rdf#tri27
subj: http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2119instance
pred: http://xmlns.com/foaf/0.1/name
obj (Literal): Peter Bungert

uri: http://www.aifb.uni-karlsruhe.de/Personen/viewPersonFOAF/foaf_2069.rdf#tri132
sub: http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2062instance
pred: http://xmlns.com/foaf/0.1/name
obj (Literal): Peter Weiß
```

These triples repeated themselves in different files (with different URIs) and so a lot of duplicate data has been indexed by the system. The user can search for the homepages of Ph.D. students named `peter` by posing the query, `getBindings([<phdstudent>::<peter> <homepage> ?x])`, which returns the following results:

```
http://www.aifb.uni-karlsruhe.de/WBS/pha/
http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/WBS
http://www.aifb.uni-karlsruhe.de/Personen/viewPerson?id_db=2023
http://www.aifb.uni-karlsruhe.de/Personen/viewPerson?id_db=2119
http://www.aifb.uni-karlsruhe.de/Personen/viewPerson?id_db=2062
```
The above URIs are, apparently, home pages of the above three individuals. The interesting thing is that all of the URIs except the first one points to a German page (whose content has not been indexed by our system). In order to pose queries such as *get homepages of Ph.D. students named pete r which talk about “semantic grid”* which translates into *getDocsByBindingsAndContent([<phdstudent>::<peter> <homepage> ?x] “semantic grid”)*, we must convey to the system that the German version of the page should be treated “same as” the English version. Now that the user has the names, she can use the names to query the system. The query *<peter haase>* retrieves these URIs:

- http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2023instance
- http://www.aifb.uni-karlsruhe.de/Personen/viewPerson?id_db=2023
- http://www.aifb.uni-karlsruhe.de/Personen/viewPersonenglish?id_db=2023
- http://www.aifb.uni-karlsruhe.de/Publikationen/viewPublikationenPersonOWL/id2023.owl
- http://www.aifb.uni-karlsruhe.de/Personen/viewPersonOWL/id2023.owl

These URIs are a mix of HTML (second and third URIs) and OWL documents. The second URI is the homepage of the individual named *Peter Haase*. As discussed previously, the page is almost synonymous with the individual and so the pages that link to (*linksTo*) this page must be, arguably, relevant to the individual. The query *getLinkingNodes (http://www.aifb.uni-karlsruhe.de/Personen/viewPerson?id_db=2023)* retrieves a set of RDF and HTML documents most of which are pages of projects on which Peter Haase is working. Some of these results are shown below:

- http://www.aifb.uni-karlsruhe.de/Personen/viewPersonFOAF/foaf_2023.rdf
- http://www.aifb.uni-karlsruhe.de/Personen/Projekte/viewProjektenenglish?id_db=78
- http://www.aifb.uni-karlsruhe.de/Personen/Projekte/viewProjektenenglish?id_db=80
- http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenenglish?id_db=51
- http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenenglish?id_db=71
- http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenenglish?id_db=81
- http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenenglish?id_db=42
- http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenenglish?id_db=54

The user can query for publications by Peter Haase that have the word “semantic” in the title by composing the query:
getBindings([<peter haase> <publication> ?x] [?x <title> <semantic>])

which retrieves the following URIs:

http://www.aifb.uni-karlsruhe.de/Publikationen/viewPublikationOWL/id399instance
http://www.aifb.uni-karlsruhe.de/Publikationen/viewPublikationOWL/id449instance
http://www.aifb.uni-karlsruhe.de/Publikationen/viewPublikationOWL/id748instance
http://www.aifb.uni-karlsruhe.de/Publikationen/viewPublikationOWL/id1003instance

All of the above are OWL files corresponding to publications. The user can query for documents asserting the triples used to find the above bindings by using a query such as getAssertingNodes([<peter haase> <publication> ?x]). The query getDocByKeywords corresponds to straight-forward keyword search of HTML documents. Note that in all of the above queries, the user is using intuitive keywords to explore the RDF data. She is not aware of the underlying schema and hardly ever needs to know the exact URIs of the resources. The user however is required to have an idea of the underlying model. The idea is to retrieve data and document nodes from the same unified whole. This will especially be useful when dealing with documents that have both text and semantic markup. Such documents can be indexed using URIs, triples and text, and the getLinkingNodes and getAssertingNodes will play a major role in their retrieval.

The above results show that HQL can be used to query and retrieve RDF and HTML documents. Comparing RDF document retrieval with other search systems is pointless unless they provide a querying mechanism to retrieve RDF documents. Most of the search systems do not even handle XML let alone RDF.

7.2.2 Quantitative analysis

SITAR allows informed users to use the wide range of queries that HQL offers to query and retrieve documents. But lay users can retrieve documents using the regular keyword
query language with minor enhancements. For example, a person searching for Jaguar the car could use the wordset pair <car>::<jaguar> to disambiguate the query. In order to compare the effectiveness of this type of document retrieval with regular document retrieval, we performed the following experiment.

As mentioned earlier, the SEAL dataset has a URI and a homepage corresponding to each entity. The URI (as opposed to the URL of the homepage) participates in the RDF triples associated with the entity. As discussed earlier, the URL of the homepage can be viewed as representing the entity itself. So we artificially created triples to the effect that the URI representing the entity is equivalent to the URL of its homepage. Whenever a URI is retrieved in response to a query, all its equivalent URIs are added to the result set too. As discussed at the beginning of the section, there are two different ways to add semantic information to a webpage. We can either involve it in a triple or we can make it linksTo a URI that participates in a triple. By involving the homepage URL in a triple, we have added semantic information to all the webpages that linksTo the homepage (this includes those pages that simply have a hyperlink to this page).

We formulated ten different simple HQL queries and picked 50 XHTML documents from the SEAL dataset. The documents were picked semi-randomly in the sense that we picked documents using general keyword searches based upon each of the queries. We had to do this to make sure we had enough documents in the mix that are relevant to the queries. We then manually examined each of these documents and recorded which of the documents are relevant to each query. So in effect, we created a table which had the query number on the left side and the number of the relevant documents on the right side.
We then ran each of the HQL queries against the dataset and, not surprisingly, found that both the *recall* and *precision* are 100% for each of the queries. The SEAL document set is such that a hyperlink is provided to the homepage of the entity whenever it is mentioned in a document. But what is interesting is the performance of plain text search for the same queries. We indexed the same set of documents using Lucene and tried to translate the queries into plain keyword queries. Table 1 shows the HQL queries and the equivalent keyword queries and the resulting *recall* and *precision* values.

**Table 1: Results**

<table>
<thead>
<tr>
<th>HQL Query</th>
<th>Keyword Query</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;professor&gt;:::</td>
<td>Andreas</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>&lt;andreas&gt;</td>
<td>Professor Andreas</td>
<td>0.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Professor Andreas”</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Professor AND Andreas</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>&lt;phdstudent&gt;:::</td>
<td>Patrick</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>&lt;Patrick&gt;</td>
<td>PhD student Patrick</td>
<td>0.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PhD AND student AND Patrick</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“PhD student” AND Patrick</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“PhD student” Patrick</td>
<td>0.43</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3</td>
<td>&lt;lecturer&gt;::&lt;peter&gt;</td>
<td>Peter</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lecturer Peter</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lecturer AND Peter</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Lecturer Peter”</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>&lt;research group&gt;::&lt;algorithms&gt;</td>
<td>Algorithms</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“research group” AND “algorithms”</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Research group” algorithms</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Research group algorithms</td>
<td>0.04</td>
</tr>
<tr>
<td>5</td>
<td>&lt;research group&gt;::&lt;knowledge management&gt;</td>
<td>Knowledge Management</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“knowledge management”</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“research group” AND</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>&lt;project&gt;::&lt;kaon&gt;</td>
<td>Kaon</td>
<td>0.88</td>
</tr>
<tr>
<td>---</td>
<td>------------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>Project Kaon</td>
<td></td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td>Project AND Kaon</td>
<td></td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>“project Kaon”</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>&lt;project&gt;::&lt;mike&gt;</td>
<td>Mike</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>Project Mike</td>
<td></td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>Project AND Mike</td>
<td></td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>“Project Mike”</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>&lt;research topic&gt;::&lt;organic computing&gt;</td>
<td>Organic Computing</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>“Organic Computing”</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>&lt;research topic&gt;::&lt;knowledge representation&gt;</td>
<td>Knowledge Management</td>
<td>0.27</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------</td>
<td>----------------------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>“research topic” AND “knowledge management”</td>
<td>“Knowledge Management”</td>
<td><strong>0.39</strong></td>
</tr>
<tr>
<td></td>
<td>“research topic” AND “knowledge management”</td>
<td>“knowledge management”</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>&lt;research topic&gt;::&lt;Digital libraries&gt;</td>
<td>Digital libraries</td>
<td><strong>0.33</strong></td>
</tr>
<tr>
<td></td>
<td>“Digital libraries”</td>
<td>“Digital Libraries”</td>
<td><strong>0.57</strong></td>
</tr>
<tr>
<td></td>
<td>“research topic” AND “Digital Libraries”</td>
<td>“digital libraries”</td>
<td>0</td>
</tr>
</tbody>
</table>

| “research topic” “organic computing” | 0.4 | 1 |
| “research topic” AND “organic computing” | 0 | 0 |
Consider the first query where the user’s information need is to retrieve documents about a professor whose name is Andreas. Note that there are documents talking about several different people named Andreas in the dataset. This need can be translated into plain keyword search in several different ways. The first keyword query Andreas is a very general one. It retrieves all the documents in which the term Andreas occurs and hence the recall value is very high (10/11 = \textbf{0.91}). But it suffers from poor precision (10/24 = \textbf{0.42}) since any document that mentions Andreas (irrespective of whether or not it is the professor we want) is retrieved in response. The second query Professor Andreas has an implicit OR between Professor and Andreas. This query searches for documents that contain either the term Professor or the term Andreas and hence the number of returned documents is more than it was in the previous case. The recall in this case remains the same (10/11 = \textbf{0.91}) while the precision suffers further (10/26 = \textbf{0.38}). The third query, “Professor Andreas”, encloses the previous query in quotes. This essentially means that the system retrieves pages where the two terms occur in sequence. This turned out to be overly discriminating and returned zero documents. The fourth query Professor AND Andreas essentially retrieves documents that have both the keywords Professor and Andreas (compare with query Professor Andreas). This again turned out to be overly discriminating. The precision was very high in this case but that’s because only one document was returned which happened to be a relevant one. As a consequence the recall was very poor (1/11 = \textbf{0.09}).

All the other queries follow a similar trend. The queries show the limitation of the standard keyword query language when it comes to conveying the user information need to the system. Even when the user knows what she is looking for, she is unable to
unambiguously convey it to the system. The above table also shows that the precision is usually very low. When precision is high, it is only because very few relevant documents were retrieved and hence recall suffers.

The experiment shows that documents can be annotated simply by adding hyperlinks from the document to a node which participates in a triple. It also shows that when such semantic annotations are present, the precision is very high compared to the standard keyword based search system provided the user can express disambiguation information to the system.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobs</td>
<td></td>
</tr>
<tr>
<td>Cooperations</td>
<td></td>
</tr>
<tr>
<td>Contact</td>
<td></td>
</tr>
<tr>
<td>AIFB Home</td>
<td></td>
</tr>
<tr>
<td>Status:</td>
<td>active</td>
</tr>
<tr>
<td>Next milestone:</td>
<td>11.08.2007</td>
</tr>
<tr>
<td>Description:</td>
<td>Dierolf, Henning, Eichhorn, Daniel, Oberweis, Andreas, Trunko, Ralf</td>
</tr>
<tr>
<td>Staff:</td>
<td></td>
</tr>
<tr>
<td>Forerunner project:</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 8: SEAL Hyperlinks**

But it should be noted that in the dataset every page that talked about an entity (such as a person named Andreas) has a hyperlink to the homepage of the person (Figure 8) Since this homepage participates in a triple, the page has semantic annotations. But if such annotations do not exist, the user can always fall back on plain keyword search and no harm is done. Note that the user can also combine wordset pairs with plain keywords and pose queries such as `getDocsByIndexAndContent (<professor>::<andreas>`
semantic grid) which retrieves pages talking about professor Andreas and semantic grid. These hybrid queries combine the semantic search with plain keyword based search.

The SEAL data proved to be invaluable for our experiments. But in order to illustrate how recall can improve in cases where synonymy is involved, we experimented with Wikipedia webpages. Wikipedia is another place where an entity has its own homepage. For example, the page [http://en.wikipedia.org/wiki/Nalanda](http://en.wikipedia.org/wiki/Nalanda) contains information about the ancient university Nalanda. This URL can be viewed as the unique identifier of this entity.

---

**Figure 9: Wikipedia Muhammad Ali disambiguation page**

The entity of interest to us was the boxer Muhammad Ali whose wiki page can be found at the URL [http://en.wikipedia.org/wiki/Muhammad_Ali](http://en.wikipedia.org/wiki/Muhammad_Ali). To perform our experiment, we manually collected 12 Wikipedia documents that talked about the boxer Muhammad Ali. One of these 12 documents was a Wikipedia disambiguation page for the name *Muhammad Ali*. A disambiguation page essentially lists the various individuals named *Muhammad Ali* and points to the URLs of the Wiki pages corresponding to those
individuals (see Figure 9). Among these 12 documents was also the disambiguation page for Cassius Clay. Note that the disambiguation page for Cassius Clay has an entry pointing to http://en.wikipedia.org/wiki/Muhammad_Ali with anchor text Cassius Marcellus Clay Jr.

![Cassius Clay (disambiguation)](image)

Figure 10: Wikipedia Cassius Clay disambiguation page

To these 12 documents, we added 8 documents which were Wikipedia pages about other individuals named Muhammad Ali. We then indexed the 20 documents in our system and manually added the following triple to the index:

Subject: http://en.wikipedia.org/wiki/Muhammad_Ali
Predicate: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Object: http://www.system.org/Boxer

We then ran the queries listed in Table 2 on the system and computed precision and recall values. As can be seen from Table 2, the precision and recall values for the wordset <Muhammad Ali> are no different from those of the keyword query Muhammad Ali. These queries essentially retrieve all documents that have the terms Muhammad and Ali in them. Also, as expected and as shown in the previous experiment, the query <Boxer>::<Muhammad Ali> fares much better and its recall and precision is 100%. But for the query Cassius Clay recall and precision values are very low whereas the query <Cassius Clay> has 100% precision and recall. This is because the UIW of the node representing Muhammad Ali gets the terms Cassius and Clay from the anchor text in the
Cassius Clay disambiguation page. Note that had there been several different Cassius Clays in the document set, the query <Cassius Clay> would have retrieved all of them just like the query <Muhammad Ali> did.

<table>
<thead>
<tr>
<th>Table 2: Wikipedia results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
</tr>
<tr>
<td>--------------------------------</td>
</tr>
<tr>
<td><strong>Keyword Query</strong></td>
</tr>
<tr>
<td>Muhammad Ali</td>
</tr>
<tr>
<td>Boxer AND Muhammad AND Ali</td>
</tr>
<tr>
<td>Cassius Clay</td>
</tr>
<tr>
<td><strong>HQL Query</strong></td>
</tr>
<tr>
<td>&lt;Muhammad Ali&gt;</td>
</tr>
<tr>
<td>&lt;Boxer&gt;::&lt;Muhammad Ali&gt;</td>
</tr>
<tr>
<td>&lt;Cassius Clay&gt;</td>
</tr>
<tr>
<td>&lt;Boxer&gt;::&lt;Cassius Clay&gt;</td>
</tr>
</tbody>
</table>

The experiment once again shows how existing HTML pages can be annotated simply by involving the URLs of the homepages of entities in triples. But more importantly, the
experiment shows how recall can improve in cases where an entity has an alias. The recall improves because a node is indexed by text that is obtained from its neighboring nodes (such as anchor text). Suppose we add the following triple to the system:

```plaintext
Subject : http://en.wikipedia.org/wiki/Muhammad_Ali
Predicate : http://www.system.org/PlaceOfBirth
Object : Louisville
```

Now the user can pose the query `[@Cassius Clay@ <place of birth> ?x]` and get an answer even though the terms Cassius Clay do not appear in the data anywhere.

The above queries are all the kind of queries that can be easily composed by laymen. Informed users can compose more advanced queries such as

```plaintext
getLinkingNodes(getBindings([?x <type> <mammal>]))
```

to do a broadening search resulting in higher recall. The result set of the above query will contain documents talking about cats, dogs, humans, horses, etc., even though these terms don’t appear in the query or the document. In order to test this we ran the query `getLinkingNodes(getBindings([?x <type> <Person>::<Boxer>]))` which then returned all the documents talking about Muhammad Ali the boxer thereby resulting in 100% recall. In contrast, the keyword query Boxer was able to retrieve only seven documents out of twelve resulting in a recall of 58%. In the future, we will extend the query language to enable even lay persons compose such queries using simple constructs. The user can pose a query such as `@Professor@::<?x>` to retrieve all entities of type Professor.
8 Conclusion

The Semantic Web has been evolving into a web of data separate from the Hypertext Web of documents. Data Retrieval techniques are widely used to retrieve data from the SW whereas Information Retrieval techniques are used to retrieve documents from the Hypertext Web. Taking a unified view of the two webs can enable the exploitation of their interconnections resulting in better data and document retrieval. In this dissertation, we have presented the Unified Web model that encodes the two webs and their interconnections. We have then showed how the model paves the way for hybrid retrieval techniques. We have formulated and implemented the Hybrid Query Language and have showed that both data and document retrieval benefit from the unification.

We have compared our system SITAR with the standard search engine Lucene and, using data from real and live websites, have shown how document retrieval can improve in terms of precision and recall. More specifically, we have shown how hyperlinks can be used as semantic annotations thereby enabling semantic reasoning to be used to improve precision. It is worth mentioning here that one of the most important contributions of this dissertation is the idea of the wordset pair which shows how reasoning can be combined with keyword-based search. The standard search engines do not have a serious problem as far as recall is concerned. But we have shown how even recall is better in special cases such as when synonymy is involved. Recall improves in these cases because the synonymous terms are contributed to a URI’s index words by different documents. For example, in Chapter 7 we have shown how the URI http://en.wikipedia.org/wiki/Muhammad_Ali ends up getting indexed by both Muhammad
Ali and Cassius Clay. Recall can also improve where the user is performing a broadening search such as obtain all documents about mammals though in this case the user is currently required to be conversant with the underlying RDF model as opposed to the previous cases where the user can compose queries which are but a minor extension of the usual keyword query language. One more additional benefit of our approach is that RDF ontology documents and hybrid documents can also be retrieved in addition to the HTML documents.

As far as data retrieval is concerned, by its very definition, data retrieval is not something that can be “improved” – the answer set to a given SQL query, given the same data, should always be the same no matter how the database is implemented. Any “improvement” in data retrieval can only be in terms of efficiency, consistency of transactions etc., which are not of concern to us. What we mean by an enhancement in data retrieval is the following: The user can use keyword-based querying of the RDF graph which does not require him to be fully conversant with the schema of the underlying RDF data. This keyword-based querying of RDF graph benefits from situating the RDF data in the Hypertext Web because data nodes can now get their index words from the document nodes in their neighborhood. In the Muhammad Ali example, the user can compose a query such as $[?x <date of birth> <Cassius Clay>]$ and obtain the date of birth of Muhammad Ali even though the underlying RDF data does not contain the Cassius Clay information. Even though these sort of queries can at present be composed only by informed users who have an understanding of the underlying RDF model they are a step in the right direction. These kinds of technologies pave the way for future keyword
based question answering systems where a lay user can go to a website and type a question such as *get me the phone number of Professor John Smith* and get the answer.

### 8.1 Future Research

The following are some of the directions our research may take in the future:

1. One of the drawbacks of our system is that the user is expected to know the RDF model in order to retrieve data from the system. In the future, we would like to study if it is possible to map a keyword query into a HQL query or if it is possible to provide intuitive GUI to help lay users compose sophisticated HQL queries.

2. The concept of trust has not been incorporated into the system. We would like to utilize the concept of trust to rank URIs.

3. When it comes to broadening searches, the user has to currently compose advanced HQL queries. We would like to provide a construct such as `<mammal>::<?x>` which the user can employ to indicate that she wants to retrieve all the instances and subclass instances of mammal. The semantics of this query needs to be studied more carefully.

4. We would like to study automatic construction of ontologies from Wikipedia portals. One of the central points of this thesis is that hyperlinks can be used as semantic annotations. If we can construct, say, a Computer Science ontology based upon Wikipedia Computer Science portal found at [http://en.wikipedia.org/wiki/Category:Computer_science](http://en.wikipedia.org/wiki/Category:Computer_science), then the researchers simply have to provide a hyperlink to the wiki page in which they are performing research. For example, a professor working on Bayesian networks can simply
provide a hyperlink from his homepage to the wikipage for Bayesian networks found at http://en.wikipedia.org/wiki/Bayesian_network. His page can then be retrieved in response to the user query for researchers on Artificial Intelligence. In other words, reasoning can be combined with hyperlink annotations to perform genuine semantic search.

5. We would like to add operators to the query language so that the users can pose queries such as retrieve all elements whose atomic weight is greater than 20.
References


[Ask] Ask, [Website], http://www.ask.com


Berkeley DB, [Website], [http://www.sleepycat.com/](http://www.sleepycat.com/)


J. Davies, R. Weeks, and U. Krohn, “QuizRDF: Search technology for the semantic web,” in *Workshop on Real World RDF and Semantic Web*


[HTML] W3C HTML Activity page, [Online resource], http://www.w3.org/MarkUp/


[Kao] KAON Tool Suite Homepage, [Website], http://kaon.semanticweb.org/

[Kao2] KAON2 Homepage, [Website], http://kaon2.semanticweb.org


[Maya] People of the Jaguar, [Webpage], http://www.civilization.ca/civil/maya/mmj01eng.html


web portal for ontology searching, ranking and classification,” In
Proceedings of the Fifth ACM Workshop on Web Information and Data

[Red] Redland RDF Application Framework, [Website], http://librdf.org/

in the Semantic Web,” in Proceedings of the 13th International World

[Rst] RDFStore Perl/C RDF storage and API, [Website],
http://rdfstore.sourceforge.net/

Semantic Web resources: the RDFStore model,” in Workshop on Semantic
Web Storage and Retrieval, Vrije Universiteit, Amsterdam, Netherlands,
2003.

[S04] A.Seaborne, "RDQL - A Query Language for RDF," [W3C Member
Submission - Online document] January 2004
http://www.w3.org/Submission/RDQL/.

[Ses] Sesame Homepage, [Website], http://www.openrdf.org

Retrieval on the Semantic Web”, in Proceedings of the 10th International
Conference on Information and Knowledge Management, November
2002.

[SWA] Semantic Web Activity page, [Online resource],
http://www.w3.org/2001/sw/.

[Swoo] Swoogle 2006 – Semantic Web Search, [Website],
http://swoogle.umbc.edu/


[UIG01] URI Planning Interest Group (W3C/IETF), “URIs, URLs, and URNs:
Clarifications and Recommendations 1.0,” [W3C Note 21 – Online

Information Retrieval Model," Proceedings of 2nd European Semantic
Web Conference (ESWC 2005), Berlin Heidelberg, New York : LNCS


Appendix A
Model description in RDF

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [ 
  <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
  <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
  <!ENTITY owl 'http://www.w3.org/2002/07/owl#'>
  <!ENTITY this 'http://www.system.org/web'>
]>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://www.system.org/web#"
  xml:base="http://www.system.org/web">

<owl:Class rdf:ID='Node'>
</owl:Class>

<rdf:Property rdf:ID='asserts'>
  <rdfs:domain rdf:resource='http://www.system.org/web#Node'/>
  <rdfs:range rdf:resource='http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement'/>
</rdf:Property>

<rdf:Property rdf:ID='hasContent'>
  <rdfs:domain rdf:resource='http://www.system.org/web#Node'/>
  <rdfs:range rdf:resource='http://www.w3.org/2000/01/rdf-schema#Literal'/>
</rdf:Property>

<rdf:Property rdf:ID='HyperlinksTo'>
  <rdfs:domain rdf:resource='http://www.system.org/web#Node'/>
  <rdfs:range rdf:resource='http://www.system.org/web#Node'/>
</rdf:Property>

<rdf:Property rdf:ID='linksTo'>
  <rdfs:domain rdf:resource='http://www.system.org/web#Node'/>
  <rdfs:range rdf:resource='http://www.system.org/web#Node'/>
</rdf:Property>

</rdf:RDF>