2007

A Unified approach To Retrieving Web Documents and Semantic Web Data

Trivikram Immaneni

Krishnaprasad Thirunarayan
Wright State University - Main Campus, t.k.prasad@wright.edu

A Unified Approach to Retrieving Web Documents and Semantic Web Data

Trivikram Immaneni[1] and Krishnaprasad Thirunarayan[1]

[1] Department of Computer Science and Engineering, Wright State University,
3640 Colonel Glenn Highway, Dayton, OH 45435, USA
{immaneni.2, t.k.prasad}@wright.edu

Abstract. The Semantic Web seems to be evolving into a property-linked web of RDF data, conceptually divorced from (but physically housed in) the hyperlinked web of HTML documents. We discuss the Unified Web model that integrates the two webs and formalizes the structure and the semantics of interconnections between them. We also discuss the Hybrid Query Language which combines the Data and Information Retrieval techniques to provide a convenient and uniform way to retrieve data and documents from the Unified Web. We present the retrieval system SITAR and some preliminary results.

Keywords: Semantic Web, Information Retrieval, Data Retrieval, Hybrid Retrieval, Unified Web, Hybrid Query Language.

1 Introduction

Semantic Web [1] is a term used to describe the family of description languages, standards, and other technologies which aim at "extending" the current web by making its content machine accessible. Since the Resource Description Framework (RDF) forms the foundation of this "extension", we can visualize the Semantic Web (SW) as a labeled graph with resources as nodes and binary predicates as edges (web of data). This is in contrast to the Hypertext Web (HW) which is a graph with resources (usually documents) as nodes and hyperlinks as edges (web of documents).

An interesting question that arises is as to where the Web documents fit into the SW and how they can be retrieved. Intuitively, since the Web documents are resources that are identified by their URIs, we can view them as nodes in the SW graph. The document *content* can be explicitly incorporated into the SW as literals. We can then use RDF query languages such as SPARQL [2], which enable RDF graph traversal and support regular expression matching of strings (literals), to retrieve the documents based upon their neighborhood as well as their content. For example, we can pose queries such as *retrieve documents authored by Tragula and contain the string "Spectrographic"*. This is classic Data Retrieval (DR).

Arguably, for this method of retrieving Web documents to have any remote chance of out-performing current Information Retrieval (IR) techniques, each and every Web document should have highly useful semantic *descriptions*. Some technologies such as RDFa [3] enable embedding of semantic markup in a HTML document. Even if such technologies gain wide usage, unless we find a way to (automatically) create

semantic descriptions of all of the existing HW documents, there will always be a large corpus of documents isolated from the SW. Another issue here is that query languages such as SPARQL require the users to have intimate knowledge of the underlying schema (exact URIs) to compose queries. The simple keyword-based interfaces that systems such as Yahoo! and Google expose to their users is another compelling reason to stick to IR techniques for retrieving Web documents. So, we seem to be better off retrieving data from the SW using DR techniques and retrieving documents from the HW using keyword-based IR techniques. In this sense, when seen from (data or document) retrieval perspective, the Semantic Web is, conceptually, a web of data that is estranged from the web of documents that is the Hypertext Web.

Our high level goal is to *view* the Semantic Web and the Hypertext Web as a unified whole and retrieve data and documents from this Unified Web (UW) [4]. This way, we can utilize the available semantic descriptions to enhance Web document retrieval and will also have the option of using the information from the (unstructured documents of) HW to improve the SW data retrieval.

The web documents can be broadly divided into the following three categories – those meant primarily for human consumption (HTML, plain text, jpg, etc.), those meant primarily for machine consumption (RDF, OWL, RDFS, etc.) and hybrid documents that are meant for both machine and human consumption (RDFa, microformats and other such technologies that allow embedding of semantic markup in HTML/XHTML documents [5]). Our goal is to facilitate the retrieval of all the above three types of documents while fully exploiting semantic markup/descriptions when available to increase retrieval effectiveness.

We want to enable lay users to retrieve human-consumable documents (first and third types) using the traditional keyword-based query mechanism (with minimal enhancements). We want to transparently use the available SW data to enhance the retrieval process. For example, if the user knows that she is looking for Jaguar the car, she should be able to communicate this disambiguating information to the system using a query such as "car::jaguar".

For more informed users, we want to provide a light-weight, keyword-based hybrid query language, that does not require knowledge of the underlying schema (exact URIs). These users should be able to use the query language to retrieve all three types of documents. That is, they should be able to (i) retrieve human-consumable documents by posing queries such as *retrieve documents authored by Tragula (the professor) and are about spectrography,* or *retrieve homepages of professors named John*, (ii) query for and retrieve SW documents (RDF, OWL, RDFS, etc.) by posing queries such as *retrieve documents that assert triples about the ventriloquist John Smith,* and (iii) retrieve hybrid documents (e.g., RDFa) by posing queries that combine the features of the above two types of queries. In addition, the users should be able to query and retrieve *data* by posing questions such as *what is professor Tragula's phone number* or *list all the elements in group 1 of the periodic table*, even in the absence of schema information.

We first describe the Unified Web model in Section 2, followed by the description of the Hybrid Query Language in Section 3. We discuss the implementation of our system SITAR and present some results in Section 4. We discuss related research in Section 5 and conclude with Section 6.

2 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*). A node is an abstract entity that is uniquely identified by its URI. 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.

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 1).

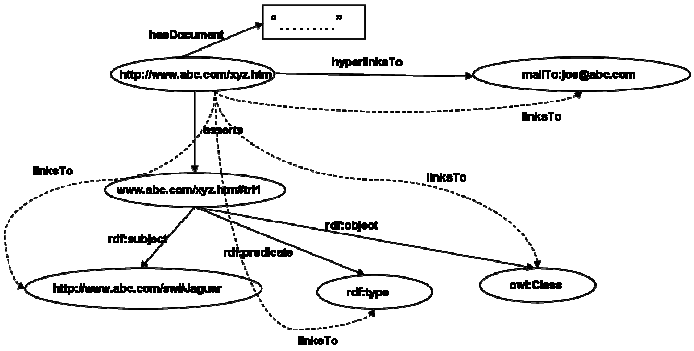

Figure 1. Relationships

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

$hyperlinksTo \subseteq DN \times NN$
$linksTo \subseteq DN \times NN$
$asserts \subseteq DN \times TN$
$hasDocument: DN \rightarrow LN$

These relations are not independent and cannot be assigned arbitrarily. They must satisfy at least the following constraints:

$\forall n \in DN, \forall m \in NN$: if [n, *hyperlinksTo*, m] $\in SE$ then [n, *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 *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 (*hyperlinksTo* and *hasDocument*) and the SW and the connections between the two (such as *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 [4]. The *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 *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 *linksTo* to view a document as being annotated by the URIs that it *linksTo* and use this information while retrieving documents.

The UW model can be specified and implemented using RDF [4]. Since all the "user triples" are present (in reified from) in the model, query languages like SPARQL can be used to retrieve the data – all we need is a straight-forward mapping of the SPARQL query for the SW to the SPARQL query for the UW.

3 Hybrid Query Language Specification

The Hybrid Query Language [4] 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:

homeURI: N → Set(URI)
externalTexts: NN → PowerSet(STRINGS)
indexWords : NN → PowerSet(STRINGS)
parameters: DN → PowerSet(STRINGS)
hasTriples: DN → PowerSet(NN x NN x NN)
hasLiteral: LN → STRINGS

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. *externalTexts* maps a node to a set of strings, possibly from its neighborhood, providing contextual information. *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. *parameters* maps a document node to a set of attribute-value strings capturing OS/Server related book-keeping information on the document. *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:

$$\forall n \in NN, \forall [n1, n2, n3] \in NN \times NN \times NN:$$
$$[n1, n2, n3] \in hasTriples\ (n) \quad \text{only if}$$
$$[n, linksTo, n1] \in SE \ \wedge\ [n, linksTo, n2] \in SE \ \wedge\ [n, linksTo, n3] \in SE$$

$$\forall n \in N, \forall [n1, n2, n3] \in NN \times NN \times NN:$$
$$[n1, n2, n3] \in hasTriples\ (n) \quad \text{if and only if}$$
$$\exists\ tn \in TN : [n, asserts, tn] \wedge [tn, rdf{:}subject, n1] \in SE$$
$$\wedge [tn, rdf{:}predicate, n2] \in SE \wedge [tn, rdf{:}object, n3] \in SE$$

For convenience, we abuse the language and say that n1, n2, and n3 *appear in* tn in the context of the reification constraint.

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 *hasDocument*, *hasLiteral*, etc. and from triples using *rdf:subject*, *rdf:predicate*, and *rdf:object*, etc.

TopLevelQuery ::= Nodes-ref | Triples-ref | ...

QUERY: *Nodes-ref ::= u*, where $u \in Set(URI)$.

ANSWER: $Result(u) = \{ n \in N \mid HomeURI(n) = u \}$
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 $ss \in PowerSet(STRINGS)$.
ANSWER: $Result(ss) = \{ n \text{ in } N \mid ss \subseteq IndexWords(n) \}$
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 $pp, ss \in PowerSet(STRINGS)$.
ANSWER: $Result(pp::ss) = \{ n \in N \mid ss \subseteq IndexWords(n) \wedge \exists m : n \text{ ISA } m \wedge pp \subseteq IndexWords(m) \}$
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: *Triples-ref* ::= u, where $u \in Set(URI)$.
ANSWER: $Result(u) = \{ n \in TN \mid HomeURI(n) = u \}$
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/Personen/viewPersonFOAF/foaf_80.rdf#tri52*

QUERY: *Triples -ref* ::= *Single-Var-Triples-ref*
Single-Var-Triples -ref ::= [?var *Nodes-ref Nodes-ref*]
Single-Var-Triples -ref ::= [*Nodes-ref* ?var *Nodes-ref*]
Single-Var-Triples –ref ::= [*Nodes-ref Nodes-ref* ?var]
where *?var* is a variable.
ANSWER: $Result([?var\ Nodes\text{-}ref1\ Nodes\text{-}ref2]) =$
$\{ t \in TN \mid n1 \in Result(Nodes\text{-}ref1) \wedge n2 \in Result(Nodes\text{-}ref2) \wedge \exists m \in N : [m, asserts, t] \wedge [t, rdf:predicate, n1] \wedge [t, rdf:object, n2] \}$
Similarly, for the other two cases.
SEMANTICS: The *part triple query* [?var *Nodes-ref1 Nodes-ref2*] returns the set of (system generated) triple nodes that are 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> <atomic weight> ?x]*

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

SEMANTICS: The *full triple query* [*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: *Triples-ref* ::= *Double-Var-Triples-ref*
Double-Var-Triples-ref ::= [?var, ?var, *Nodes-ref*]
Double-Var-Triples-ref ::= [?var, *Nodes-ref*, ?var]
Double-Var-Triples-ref ::= [*Nodes-ref*, ?var, ?var]
where ?var is a variable.
ANSWER: *Result*([?var, ?var, *Nodes-ref*]) =
$\{ t \in TN \mid m \in Result(Nodes\text{-}ref) \wedge [t, rdf{:}object, m] \wedge \exists n \in N : [n, asserts, t] \}$
Similarly, for the other two cases.
SEMANTICS: The *part triple to triples query* [?var ?var *Nodes-ref*] returns the set of (system generated) triple nodes that are related to 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: *Nodes-ref* ::= *Nodes-ref AND Nodes-ref*
Nodes-ref ::= *Nodes-ref OR Nodes-ref*
Triples-ref ::= *Triples-ref AND Triples-ref*
Triples-ref ::= *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.

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

QUERY: *Nodes-ref* ::= *getAllTriples*(*Nodes-ref*)
ANSWER: *Result*(*getAllTriples*(*Nodes-ref*)) =
$\{ t \in TN \mid n \in Result(Nodes\text{-}ref) \wedge n \text{ appears in } t \wedge \exists m \in N : [m, asserts, t] \}$
SEMANTICS: This query retrieves the (system generated) triple nodes in which the queried node URI appears.
EXAMPLE: *getAllTriples(http://www.daml.org/2003/01/periodictable/PeriodicTable#group_11)*

QUERY: *Nodes-ref* ::= *getLinkingNodes*(*Nodes-ref*)
ANSWER: *Result*(*getLinkingNodes*(*Nodes-ref*)) =
$Result(Nodes\text{-}ref) \cup \{ m \in N \mid \exists n \in Result(Nodes\text{-}ref) : [m, linksTo, n] \}$
SEMANTICS: This query retrieves the nodes corresponding to *Nodes-ref* and the document nodes containing references to the nodes corresponding to *Nodes-ref*. Effectively, nodes and their neighborhoods are retrieved.
EXAMPLE: *getLinkingNodes(http://www.aifb.uni-karlsruhe.de/Personen/viewPerson?id_db=2023)*

QUERY: *Nodes-ref* ::= *getAssertingNodes*(*Triples-ref*)
ANSWER: *Result*(*getAssertingNodes*(*Triples-ref*)) =
{ m ∈ *DN* | ∃t ∈ *Results*(*Triples-ref*) : [m, *asserts*, t] }
SEMANTICS: This query retrieves document nodes containing the triples.
EXAMPLE: *getAssertingNodes([<peter haase> <publication> ?x])*

QUERY: *Nodes-ref* ::= *getDocsByKeywords*(*ss*), where *ss* ∈ *PowerSet(STRINGS)*
ANSWER: *Result*(*getDocsByKeywords*(*kws*)) =
{ m ∈ *DN* | *hasDocument*(m) = dt ∧ *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. *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, *match* (kws, dt) = ∀w ∈ kws: *match*(w, dt), but it is not required.

QUERY: *Nodes-ref* ::= *getLiteralsByKeywords* (*ss*),
where *ss* ∈ *PowerSet*(*STRINGS*)
ANSWER: *Result*(*getLiteralsByKeywords*(kws)) =
{ m ∈ *LN* | *hasLiteral*(m) = dt ∧ *match*(kws, dt) }
SEMANTICS: This is analogous to the above query customized for literal nodes.
EXAMPLE: *getLiteralsByKeywords(semantic grid)*

3.2 Further Queries for Retrieving Documents

QUERY: *getDocsByContent*: *PowerSet*(*STRINGS*) → PowerSet(DN)
ABBREVIATION FOR: *getDocsByContent*(kws) =
getLinkingNodes(*getDocsByKeywords*(kws))
where kws ∈ *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" [6], assisting both navigational searches and research searches [7].

QUERY: *getDocsByIndexOrContent*: *PowerSet*(*STRINGS*) → *PowerSet*(*DN*)
ABBREVIATION FOR: *getDocsByIndexOrContent* (kws) =
$$getDocsByKeywords(kws) \vee \bigvee_{kw \in kws} getLinkingNodes(kw)$$
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*:
Nodes-ref x *PowerSet*(*STRINGS*) → *PowerSet*(*DN*)

ABBREVIATION FOR: *getDocsByIndexAndContent* (nr, kws) =
getLinkingNodes(*Result*(nr)) ∧ *getDocsByKeywords*(kws)
where nr ∈ *Nodes-ref*, kws ∈ *PowerSet*(*STRING*S)
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.

QUERY: *getDocsByTriplesAndContent*:
Triples-ref x *PowerSet*(*STRINGS*) → *PowerSet*(*DN*)
ABBREVIATION FOR: *getDocsByTriplesAndContent*(tr, kws) =
getAssertingNodes(tr) ∧ *getDocsByKeywords*(kws)
where tr ∈ *Triples-ref*, kws ∈ *PowerSet*(*STRINGS*)
SEMANTICS: This query retrieves (semantic web) document nodes that match the keywords and contain the referenced triples.

QUERY: *Single-Var-Triples-list* ::= *Single-Var-Triples-ref*
Single-Var-Triples-list ::= *Single-Var-Triples-ref*
Single-Var-Triples-list
Nodes-ref ::= *getBindings*(*Single-Var-Triples-list*)

QUERY1: *Nodes-ref* ::= *getBindings*([?var *Nodes-ref Nodes-ref*])
ANSWER: *Result*(*getBindings*([?var *Nodes-ref1 Nodes-ref2*]))
= { n ∈ N | n1 ∈ *Result*(*Nodes-ref1*) ∧ n2 ∈ *Result*(*Nodes-ref2*)
∧ ∃m ∈ N : [m, *asserts*, t] ∧ [t, *rdf:subject*, n]
∧ [t, *rdf:predicate*, n1] ∧ [t, *rdf:object*, n2]}
Similarly, for the other two cases.

QUERY2: *Nodes-ref* ::= *getBindings*(*Single-Var-Triples-ref*
Single-Var-Triples-list)
ANSWER: *Result*(*getBindings*(*Single-Var-Triples-ref Single-Var-Triples-list*)) =
Result(*getBindings*(*Single-Var-Triples-ref*)) ∩
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*:
Single-Var-Triples-list x *PowerSet*(*STRINGS*) → *PowerSet*(*DN*)

ABBREVIATION FOR: *getDocsByBindingsAndContent*(vtl, kws) =
getBindings (vtl) ∧ *getDocsByKeywords*(kws)
where vtl ∈ *Single-Var-Triples-list*,
kws ∈ *PowerSet*(*STRINGS*)

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

EXAMPLE: *getDocsByBindingsAndContent([<phdstudent>::<peter> <homepage> ?x] "Semantic Grid")*

4 Implementation and Results

We have implemented an Apache Lucene [8] based retrieval system called SITAR (Semantic InformaTion Analysis and Retrieval system) based upon our model. The system can currently index HTML and RDF/OWL files in addition to RDF data. At present, the system does not support pdf, doc files etc. (we index their URIs but their content is not being analyzed).

Evaluating such a hybrid system is an extremely tricky process. The system has DR components (triple matching) which render the precision and recall criteria irrelevant. But at the same time, the system also has IR components such as keyword based retrieval of documents in which case precision and recall become important. In order to evaluate the system in terms of precision and recall, we would need a standard data set (such as MEDLINE dataset) which has documents, their semantic descriptions, some queries, and results of those queries (adjudged to be relevant by human experts). We are still looking for such data sets. Here, we present qualitative results obtained by experimenting with the invaluable AIFB SEAL [9] data.

The AIFB SEAL website has human-consumable XHTML documents (in English and German) along-side OWL documents. Some of the XHTML documents have explicit semantic descriptions (in the OWL documents). We crawled the SEAL website looking only for English versions of web pages and RDF/OWL files using heuristics. The crawler collected a total of 1665 files. Of these, we chose to deliberately ignore some large OWL files (multiple copies of the same file with each copy identified by a different URI) to simplify matters. Our system uses the CyberNeko HTML parser [10] to parse HTML documents and the Jena ARP [11] parser to parse RDF documents. The ARP parser could not parse some of the RDF documents - possibly because of problems with container elements. In the end, a total of 1455 (610 RDF files and 845 XHTML files) were successfully parsed and indexed. 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.

Every URI 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* (or *hyperlinksTo*) as well as the words that are extracted from the URIs. 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 as mentioned before, we are not experimenting with RDFa documents at present.

A user can use the HQL (Hybrid Query Language) described in the previous section to query for data and documents. A user searching for information about a person named *peter* can pose the query *<peter>*. This query, 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 apparently 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 semantic web. 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). So, we cannot pose queries such

as *get homepages of Ph.D. students named peter which talk about "semantic grid"* which translates into *getDocsByBindingsAndContent([<phdstudent>::<peter> <homepage> ?x] "semantic grid")* , unless we can 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 the following 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. It is almost synonymous with the individual [4] 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/viewPersonDC/en/dc_2023.rdf
http://www.aifb.uni-karlsruhe.de/Personen/viewPersonFOAF/foaf_2023.rdf
http://www.aifb.uni-karlsruhe.de/Personen/Projekte/viewProjektenglish?id_db=78
http://www.aifb.uni-karlsruhe.de/Personen/Projekte/viewProjektenglish?id_db=80
http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenglish?id_db=51
http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenglish?id_db=71
http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenglish?id_db=81
http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenglish?id_db=42
http://www.aifb.uni-karlsruhe.de/Forschungsgruppen/Projekte/viewProjektenglish?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. The query *getDocByKeywords(peter haase)* retrieves 251 HTML documents. A Google search for "*peter haase*" retrieves 325 documents (with omitted results) on the AIFB website. But note that we are not indexing all the AIFB web pages and that we are completely ignoring PDF documents and the like.

SITAR indexes and retrieves RDF files too. In other words, SITAR aims at treating HTML and RDF files with equal importance. SITAR allows users to simply enter a

set of keywords which is then automatically plugged into the query *getDocsByIndexOrContent*. So, the query *peter haase* returns 299 documents which are a mix of HTML and OWL documents (it also retrieves a PDF document URI).

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. As can be imagined, this will especially be useful when dealing with those 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 retrieving those documents. We are currently looking for an RDFa like dataset to test this.

5 Related Research

Storing and retrieving RDF data is an area of research that has been well explored by researchers in the recent past [12,13,14,15]. Retrieving RDF data is typically viewed as a data retrieval problem and, not surprisingly, most of the query languages have the SQL flavor [14]. When seen purely from the perspective of querying the RDF data, HQL is unique because 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.

As far as document retrieval is concerned, there are several retrieval systems that retrieve documents based upon their annotations/descriptions [6,16,17,18,19,20,21], 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 [6] 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 SITAR, quizRDF[16] 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 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[18] 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[17] 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 [19,20,21] 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.

6 Conclusion and Future Work

We have discussed the Unified Web model that seeks to present a unified view of the SW and the HW, and the design and implementation of the Hybrid Query Language that can be used to retrieve data and documents from the UW. We have presented preliminary results obtained by experimenting with AIFB SEAL data.

HQL is a light-weight, keyword-based query language that allows the users to query and explore the RDF graph even when no schema information is available. This can then lead to composition of more involved queries using languages such as SPARQL. If, in the future, we expect lay users to pose queries such as *what is the phone number of Ph.D. student Peter Bungert,* to the Semantic Web and get back answers, query languages like HQL are a step in the right direction (though at present the user is still required to have knowledge of the RDF model).

One of the fundamental ideas behind HQL is to index a URI using a set of keywords, which is a common notion in the literature. But because we position the RDF data in a web of hypertext documents, we have the freedom to exploit information from the hypertext documents (such as the anchor text) to enrich a URI's index words. At this level, we again see natural language induced problems such as synonymy, polysemy, etc. (which only got pushed to a lower level). The resulting uncertainty necessitates ranking (not unlike what Swoogle [18] is doing). But, this is where the novel wordset pair queries such as *<phdstudent>::<peter>* enable disambiguation, stating that the user is only interested in URIs of Peter the PhD student. This, in essence, is how ontologies can help in document retrieval. And this is where the "Semantic Web enabled Information Retrieval" starts deviating from traditional IR. Otherwise, we are simply pushing the problem of keyword-based document retrieval to the level of URIs (we have simply reduced the size of a typical term vector) and there is nothing "semantic" about it – *jaguar* will retrieve both the car and animal URIs in spite of "meaningful" label-literals.

SITAR and HQL are both works in progress and are gradually evolving. The major piece of the puzzle missing from SITAR is ranking of URIs and documents. Even

though Lucene does rank URIs (SITAR stores a URI in a Lucene document that is indexed by the index words), and of course, documents, we need a ranking algorithm that is based on *linksTo* relationship among others (especially to rank RDF and hybrid files). We are currently working on a ranking algorithm and its implementation.

References

1. Semantic Web Activity page, [Webpage], http://www.w3.org/2001/sw/.
2. E. Prud'hommeaux, A. Seaborne, Eds., "SPARQL Query Language for RDF," [W3C Working Draft], October 2006, http://www.w3.org/TR/rdf-sparql-query/.
3. B. Adida, M. Birbeck, Eds., "RDFa," [W3C Working Draft], 2006, http://www.w3.org/TR/xhtml-rdfa-primer/.
4. T. Immaneni and K. Thirunarayan, "Hybrid Retrieval from the Unified Web," *Proceedings of the 22nd ACM Symposium on Applied Computing, Semantic Web and Applications Track (ACM SAC 2007)*, Seoul, Korea, March 2007.
5. K. Thirunarayan, "On Embedding Machine-Processable Semantics into Documents," in *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 7, pp. 1014-1018, July 2005.
6. J. Kleinberg, "Authoritative sources in a hyperlinked environment," *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.
7. R. Guha, R. McCool, and E. Miller, "Semantic search," in *Proceedings of the Twelfth International Conference on World Wide Web*, Budapest, Hungary, New York: ACM Press, May 2003.
8. Apache Lucene, [Webpage], http://lucene.apache.org/.
9. J. Hartmann, Y. Sure., "An Infrastructure for Scalable, Reliable Semantic Portals," IEEE Intelligent Systems 19 (3): 58-65. 2004.
10. CyberNeko HTML Parser, [Webpage], http://people.apache.org/~andyc/neko/doc/html/ .
11. Jena ARP, [Webpage], http://www.hpl.hp.com/personal/jjc/arp/.
12. D.Beckett, "SWAD-E Deliverable 10.2: Mapping Semantic Web Data with RDBMSes," [Online Document] 2003, http://www.w3.org/2001/sw/Europe/reports/ scalable_rdbms_mapping_report/
13. D. Beckett, "SWAD-Europe Deliverable 10.1: Scalability and Storage: Survey of Free Software / Open Source RDF storage systems," [Online Document] 2002, http://www.w3.org/2001/sw/Europe/reports/rdf_scalable_storage_report/.
14. J. Bailey, F. Bry, T. Furche, and S. Schaffert, "Web and Semantic Web Query Languages: A Survey," *Reasoning Web*, Eds., N. Eisinger and J. Maluszynski , Springer-Verlag, 2005.
15. P. Haase, J. Broekstra, A. Egerhart, and R. Volz, "A comparison of RDF query langauges," in *Proceedings of the Third International Semantic Web Conference*, Hiroshima, Japan, 2004.
16. J. Davies, R. Weeks, and U. Krohn, "QuizRDF: Search technology for the semantic web," *Workshop on Real World RDF and Semantic Web Applications*, 11th International World Wide Web Conference, Hawaii, USA, 2002.
17. J. Mayfield and T. Finin, "Information retrieval on the semantic web: Integrating inference and retrieval," in *Proceedings of the SIGIR 2003 Semantic Web Workshop,* 2003.
18. Li Ding et al., "Finding and Ranking Knowledge on the Semantic Web", in *Proceedings of the 4th International Semantic Web Conference*, November 2005.
19. C. Rocha, D. Schwabe, and M.P. Aragao, "A Hybrid Approach for Searching in the Semantic Web," in *Proceedings of the 13th International World Wide Web Conference*, New York, May 2004, pp. 374-383.
20. L. Zhang, Y. Yu, J. Zhou, C. Lin, Y. Yang, "An enhanced model for searching in semantic portals," in *Proceedings of the 14th International World Wide Web Conference*, Chiba, Japan, NY: ACM Press, May 2005.
21. D. Vallet, M. Fernández, and P. Castells, "An Ontology-Based Information Retrieval Model," in Proc. of 2nd European Semantic Web Conf. (ESWC 2005), Berlin Heidelberg, 2005.