Semantic Search

R. Guha
IBM Research, Almaden
rguha@us.ibm.com

Rob McCool
Knowledge Systems Lab,
Stanford
Stanford, CA, USA
robm@ksl.stanford.edu

Eric Miller
W3C/MIT
Cambridge, MA, USA
em@w3.org

ABSTRACT

Activities such as Web Services and the Semantic Web are working to create a web of distributed machine understandable data. In this paper we present an application called Semantic Search which is built on these supporting technologies and is designed to improve traditional web searching. We provide an overview of TAP, the application framework upon which the Semantic Search is built. We describe two implemented Semantic Search systems which, based on the denotation of the search query, augment traditional search results with relevant data aggregated from distributed sources. We also discuss some general issues related to searching and the Semantic Web and outline how an understanding of the semantics of the search terms can be used to provide better results.

Categories and Subject Descriptors

H.3.3 Information Search and Retrieval

General Terms

Experimentation

Keywords

Semantic Search, Semantic Web

1. THE SEMANTIC WEB

The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. It is the idea of having data on the Web defined and linked in a way that it can be used for more effective discovery, automation, integration, and reuse across various applications. In particular, the Semantic Web will contain resources corresponding not just to media objects (such as Web pages, images, audio clips, etc.) as the current Web does, but also to objects such as people, places, organizations and events. Further, the Semantic Web will contain not just a single kind of relation (the hyperlink) between resources, but many different kinds of relations between the different types of resources mentioned above.

The general ideas presented in this paper assume that the Semantic Web will contain resources with relations amoung each other. More concretely, we assume that the data on the Semantic Web is modeled as a directed labeled graph, wherein each node corresponds to a resource and each arc is labeled with a property type (also a resource). While there are several different XML based proposals for representing resources and their inter-relations on the Semantic Web, a particular system needs to make a commitment to one or more interchange formats and protocols in order to exchange this information. The system described in this paper uses the W3C's Resource Description Framework with the schema vocabulary provided by RDFS [8] as a means for describing resources and their inter-relations. SOAP [7] is used as the protocol for querying and exchanging this RDF instance data between machines.

Figures 1 and 2 show two small chunks of the Semantic Web, the first corresponding to the cellist Yo-Yo Ma and the second corresponding to Eric Miller, one of the co-authors of this paper. These two examples illustrate several salient aspects of the Semantic Web that are important to Semantic Search.

Documents vs Real World Objects: The Semantic Web is not a Web of documents, but a Web of relations between resources denoting real world objects, i.e., objects such as people, places and events. In the first example we have objects such as the city of Paris, the musician Yo-Yo Ma, an auction event, the music album Appalachian Journey, etc. In the second example, we have the person Eric Miller, the W3C Semantic Web Activity, the organization W3C, the city of Dublin, Ohio, etc.

Human vs Machine Readable Information: In Figure 2, we have a resource corresponding to Eric Miller. This is not the string *Eric Miller*, but a resource denoting a person. There are many people with the name Eric Miller. This denotes only one particular person with that name. The salient point about the Semantic Web is that it contains rich machine readable information about these resources. Compare the data in Figure 2 with Eric Miller's home page (http://www.w3.org/ People/EM/). Eric's home page contains more human readable information than Figure 2, but almost all the machine understandable parts of that Web page correspond to how it should be displayed by a browser. The data in Figure 2 on the other hand, is almost all machine understandable. It states, in a machine understandable language, that Eric is a person, that Eric works for the W3C, etc.

Relation between the HTML & Semantic Web: The Semantic Web is an extension of the current Web. As Figure 2 shows, there is a rich set of links from the nodes in the Semantic Web to HTML documents. These relations typically connect a concept in the Semantic Web with the pages that most pertain to it.

It is also possible that some of the pages in the current Web contain semantic markup ([10] [3]). However, Semantic Search

WWW2003, May 20-24, 2003, Budapest, Hungary.
ACM 1.58113-680-3/03/0005.

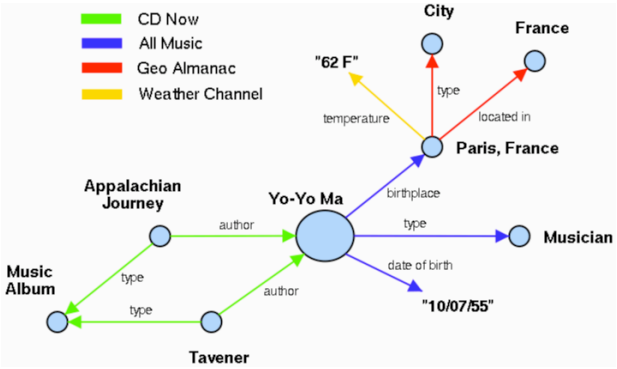

Figure 1: A segment of the Semantic Web pertaining to Yo-Yo Ma

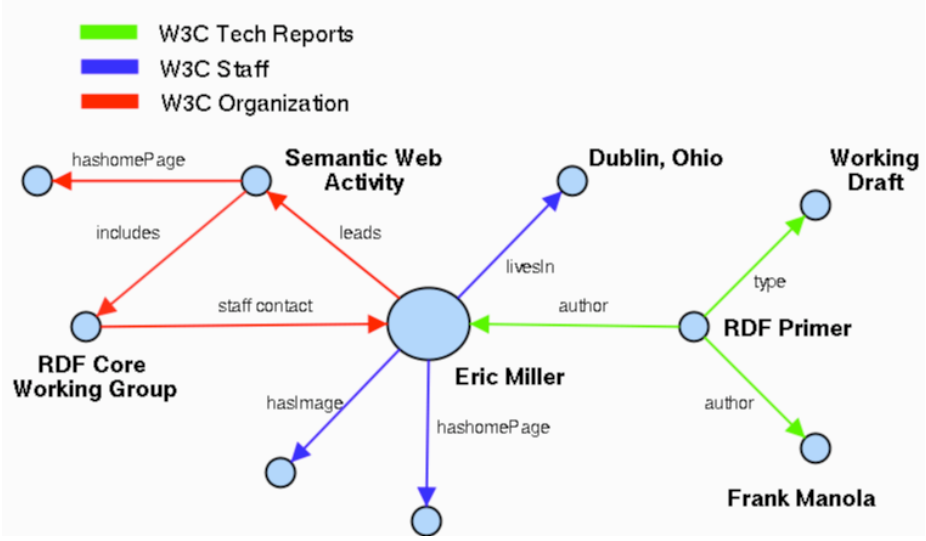

Figure 2: A segment of the Semantic Web pertaining to Eric Miller

as described in this paper does not use such markups. We assume that robots will gather such markups so that they are available on the Semantic Web.

Distributed Extensibility: Another important aspect of the Semantic Web is that different sites may contribute data about a particular resource. In the example shown in Figure 1, many different sources have data about Yo-Yo Ma and related resources. Amazon and CDNow have data about his albums, Ebay has data about auctions related to these albums, TicketMaster has data about his concert schedule, AllMusic has data about where he was born (Paris), and so on. Each of these sites can publish data about Yo-Yo Ma without getting permission from any centralized authority, i.e., they can all extend the cumulative knowledge on the Semantic Web about any resource in a distributed fashion. This *distributed extensibility* is a very important aspect of the Semantic Web.

Of course, this feature leads to problems of its own. In a world where anyone can publish anything, a lot of what gets published cannot be trusted. In the current Web, we, as humans, use our intelligence, invoking concepts of brand, who recommended what, etc. to decide whether to believe what a Web site says. Programs, on the other hand, being relatively unintelligent, do not have recourse to all these facilities to decide whether to believe the data from a new site on the Semantic Web. This is an important problem that will need to be addressed [18].

2. SEMANTIC SEARCH INTRODUCTION

As with the WWW, the growth of the Semantic Web will be driven by applications that use it. Semantic search is an application of the Semantic Web to search. Search is both one of the most popular applications on the Web and an application with significant room for improvement. We believe that the addition of explicit semantics can improve search. Semantic Search attempts to augment and improve traditional search results (based on Information Retrieval technology) by using data from the Semantic Web.

Traditional Information Retrieval (IR) technology is based almost purely on the occurrence of words in documents. Search engines like Google [9]), augment this in the context of the Web with information about the hyperlink structure of the Web. The availability of large amounts of structured, machine understandable information about a wide range of objects on the Semantic Web offers some opportunities for improving on traditional search.

Before getting into the details of how the Semantic Web can contribute to search, we need to distinguish between two very different kinds of searches.

Navigational Searches: In this class of searches, the user provides the search engine a phrase or combination of words which s/he expects to find in the documents. There is no straightforward, reasonable interpretation of these words as denoting a concept. In such cases, the user is using the search engine as a navigation tool to navigate to a particular intended document. We are not interested in this class of searches.

Research Searches: In many other cases, the user provides the search engine with a phrase which is intended to denote an object about which the user is trying to gather/research information. There is no particular document which the user knows about that s/he is trying to get to. Rather, the user is trying to locate a number of documents which together will give him/her the information s/he is trying to find. This is the class of searches we are interested in.

Example: A search query like "W3C track 2pm Panel" does not denote any concept. The user is likely just trying to find the page containing all these words. On the other hand, search queries like "Eric Miller" or "Dublin Ohio", denote a person or a place. The user is likely doing an research search on the person or place denoted by the query.

Semantic search attempts to improve the results of research searches in 2 ways.

- Traditional search results take the form of a list of documents/Web pages. We augment this list of documents with relevant data pulled out from Semantic Web. The Semantic Web based results are independent of and augment the results obtained via traditional IR techniques. So, for example, a search for Yo-Yo Ma might get augmented with his current concert schedule, his music albums, his image, etc. (see Figure 3.)

- The search phrase in Research Searches typically denotes one (or occasionally two) real-world concepts. We believe that it might be useful for the text retrieval part of the search engine to have an understanding of these concepts denoted by the search phrase. Understanding the denotation can help understand the context of the search, the activity the user is trying to perform, drive expectations on the categories of documents (pertaining to the object) likely to exist, etc.

Most of this paper is related to augmenting search results with data from the Semantic Web. In Section 6 we briefly describe the approaches we are taking to achieve the second goal.

We have built two Semantic Search systems. The first system, Activity Based Search (ABS), provides Semantic Search for a range of domains, including musicians, athletes, actors, places and products. The second system (W3C Semantic Search) is more focused and provides Semantic Search for the website of the World Wide Web Consortium (http://www.w3.org/). In both of these systems, we use the Google [9] search engine to provide the traditional text search results.

Since the Semantic Web does not yet contain much information, in addition to the Semantic Search application, we have had to build the requisite portions of the Semantic Web to provide data for the Semantic Search applications. Both the Semantic Search application and these portions of the Semantic Web have been built on top of the TAP infrastructure, which is described in the next section.

3. THE TAP INFRASTRUCTURE

TAP [17] is intended to be an infrastructure for applications on the Semantic Web. TAP provides a set of simple mechanisms for sites to publish data onto the Semantic Web and for applications to consume this data via a minimalist query interface called *GetData*. We first describe this query interface and then discuss some of the facilities provided by TAP for publishing and consuming data.

3.1 The GetData Query Interface

A number query languages have been developed for RDF ([11], [12], [14]), DAML ([2]) and more generally for semi-structured data ([13], [6]). Why do we need yet another query language?

These query languages all provide very expressive mechanisms that are are aimed at making it easy to express complex queries. Unfortunately, with such expressive query languages, it is easy to construct queries that require a lot of computational resources to process. Consequently, just as no major Website provides a SQL interface to its back end relational database, we don't expect sites, especially large ones, to use these query languages as the external interface to their data. What we need is a much lighter weight interface, one that is both easier to support and more importantly, exhibits predictable behavior. Predictable behavior is important not just for the service provider, but also the service client. A simple lightweight query system would be complementary to more complete query languages mentioned above. The lighter weight query language could be used for querying on the open, uncontrolled Web in contexts where a site might not have much control over who is issuing queries, whereas the latter is targeted at the comparatively better behaved and more predictable area behind the firewall. The lightweight query also does not preclude particular sites from aggregating data from multiple sites and providing richer query interfaces into these aggregations.

GetData is intended to be a simple query interface to network accessible data presented as directed labeled graphs. GetData is not intended to be a complete or expressive query language a la SQL, XQuery, RQL or DQL. It is intended to be very easy to build, support and use, both from the perspective of data providers and data consumers.

We want to enable machines to query remote servers for data. Since SOAP [7] provides a mechanism for performing RPC that is beginning to be widely accepted, GetData is built on top of SOAP.

GetData allows a client program to access the values of one or more properties of a resource from a graph. Each query-able graph has a URL associated with it. Each GetData query is a SOAP message addressed to that URL. The message specifies two arguments: the resource whose properties are being accessed and the properties that are being accessed.

The answer returned for a GetData query is itself a graph which contains the resource (whose properties are being queried) along with the arcs specified in the query and their respective targets /sources. Application programming interfaces hide the details of the SOAP messages and XML encoding from the programmer. So, as far as an application using the Semantic Web via GetData is concerned, it gets an API which (in an abstract syntax) looks like:

- $GetData(< resource >, < property >) \Rightarrow < value >$

Below are some examples of GetData, in an abstract syntax, operating against the graphs shown in Figures 1 and 2.

- $GetData(< Yo{-}YoMa >, birthplace), \Rightarrow < Paris, France >$
- $GetData(< Paris, France >, temperature), \Rightarrow 57F$
- $GetData(< EricMiller >, livesIn), \Rightarrow < Dublin, Ohio >$

<Yo-Yo Ma>, <Paris, France>, <Eric Miller> and <Dublin, Ohio> are references to the resources corresponding to Yo-Yo Ma, the city of Paris, France, the person Eric Miller, and the city of Dublin, Ohio. Typically, references to resources are via the URI for the resource. In this case, using the TAP KB [16], these URIs are
http://tap.stanford.edu/data/MusicianMa,_Yo-Yo
http://tap.stanford.edu/data/CityParis,_France
http://tap.stanford.edu/data/W3CPersonMiller,_Eric and
http://tap.stanford.edu/data/CityDublin,_Ohio.

Each of the above GetData queries is a SOAP message whose end point is the URL corresponding to the graph with the data.

GetData also allows reverse traversal of arcs, i.e., given a resource, a client can request for resources that are the sources of arcs with a certain label that terminate at that resource.

In addition to the core GetData interface, there are two other interfaces provided by TAP to help graph exploration. These are,

Search: The search interface takes a string and returns all the resources one of whose "title properties" contains the string (in the graph at the URL the query is addressed to). A title property is one which in an instance of the class TitleProperty in the TAP KB.

Reflection: The reflection interfaces, which are similar to the reflection interfaces provided by object oriented languages, return lists of arcs coming into and going out of a node. This is very useful for exploring a graph in the vicinity of a node without any knowledge of what might be around.

3.2 TAP Scraping

As mentioned earlier, since the Semantic Web is still rather sparse, we have had to build the requisite portions of it so as to enable Semantic Search.

All the data required for the W3C Semantic Search application came from RDF files maintained by the W3C. This data was published using TAPache (discussed in the next section). However, for the larger application dealing with musicians, athletes, places, etc., we had to construct HTML scrapers to get the data off popular sites such as Amazon, AllMusic, TicketMaster and others who have data about these objects. To facilitate this, TAP provides an infrastructure for interpreting a GetData request, mapping it to an appropriate scraping task and executing the scraping so that a client can pretend that the site offers a GetData interface to its data.

3.3 TAP Publishing

On the server side, TAP provides TAPache, an Apache HTTP server [1] module, for exposing data via the GetData interface. TAPache's goal is to make it extremely simple to publish data onto the Semantic Web. TAPache is not intended to be a high end solution for sites with large amounts of data and traffic. Flexibility and scalability are much more important than ease of use for such sites.

With the Apache HTTP server, there is a directory (typically called html or htdocs), in which one places files (html, gif, jpg, etc.) or directories containing such files, as a result of which these files are made available via http from that machine. Similarly, with

TAPache, one creates a sibling directory (typically called data) to the html directory and places RDF files in this directory. The graphs encoded in these files are automatically made accessible via the GetData interface. The URL associated with each graph is that of the file.

TAPache compiles each file (or if the files are small, aggregations of files) into memory-mappable graph structures so that queries issued against the files can be answered without incurring parsing overhead each time.

TAPache also provides a simple mechanism for aggregating the data in multiple RDF files. All the RDF files places under a certain directory can be viewed as an aggregate, available at the URL associated with the directory.

3.4 Registries and Caching

Different sites/graphs have different kinds of information (i.e., different properties) for different types of resources. Each GetData request is targeted at a particular URL corresponding to a graph which is assumed to have the data. Once we have a number of these graphs, keeping track of which graph has what data can become too complex for a client to handle.

We use a simple registry, which is available as a separate server, to keep track of which URL has values for which properties about which classes of resources. The registry can be abstracted as a simple lookup table which given a class and a property returns a list of URLs which might have values for that property for instances of that class. More complex registries, based on descriptions of objects, are also available as part of TAP. With the registry in place, a client can direct the query to the registry which then redirects the query to the appropriate site(s).

Querying many different sites dynamically can result in a high latency for each query. In order to provide reasonable performance, we cache the responses to GetData requests. This caching is part of the registry's functionality.

What we are trying to do here is in many ways similar to what the Domain Name Service (DNS) does for data about internet hosts. DNS provides a unified view of data about internet hosts sitting across millions of sites on the internet. Different kinds of servers store their host information differently. But what DNS does is to provide a uniform view into all this data so that a client can pretend that all of this data is sitting locally on their nameserver. Taking the analogy further, one could regard GetData as an extension of GetHostByName, the core query interface for DNS. Just as GetHostByName enables a client to query for one particular property (the IP address) of one class of objects (Internet Hosts), GetData enables a client to query for many different kinds of properties of many classes of objects.

TAP also provides for other functionalities such as semantic negotiation, to help reconcile different URIs used for the same object, but those topics are beyond the scope of this paper.

4. DATA SOURCES

In this section, we will describe the data sources on the Semantic Web used by the two Semantic Search systems we have built, ABS and W3C Semantic Search.

The data for ABS comes from a large number of sources. Many different sites have data about musicians, athletes, places and products. Most of these sites do not yet make their data available in a machine understandable form. To overcome this, we have written HTML scrapers which dynamically locate and convert the relevant pages on these sites into machine readable data and make them available via the GetData interface. Some of the sites we have written scrappers for include AllMusic, Ebay, Amazon, AOL Shopping, TicketMaster, People Magazine, Weather.com, Mapquest, Carpoint, Digital Cities and Walmart.com. Because of these scrapers, the Semantic Search application can pretend that the data on each of these sites is available via the GetData interface. Given the number of data sources and their distributed nature of its data sources, ABS makes extensive use of the registry and caching mechanisms provided by TAP. All these data source together yield a Semantic Web with many millions of triples.

A very important source of data for ABS is the TAP Knowledge Base [16], which is a very shallow but very broad knowledge base about a range of domains, including people (musicians, athletes, actors, politicians), organizations (companies, music groups, sports teams), places (cities, countries, states) and products. For each resource, it provides us with the *rdf:type* and *rdfs:label(s)* for the object. For the ABS system, the TAP Knowledge Base contributes about 65,000 people, organizations and places, which together cover about 17% of the searches that take place on an average day on the ODP [4] Website.

In contrast to ABS data for W3C Semantic Search comes from a relatively small number of sources, all of which are internal to the W3C. There is also much greater uniformity in the kind of data available about each object. We have experimented with two versions of W3C Semantic Search, one in which different portions of the data are distributed across three geographically dispersed machines (California, Ohio, and Boston) and another in which all the data is available on a single machine. As would be expected, the latter case provides better performance.

W3C Semantic Search uses five different data sources, which together cover the following kinds of objects.

People: This includes W3C staff and authors of various W3C documents. The data sources include each staff members contact, title and responsibilities.

W3C Activities: Each activity is related to the W3C people.

Working Groups and other Committees: Each of these is related to the activity and staff

Documents: This includes recommendations, working drafts, W3C notes, etc. Each of these is related to the working groups and activities which produced them.

News: W3C Semantic Search incorporates data from a number of RSS [5] news feeds about various newsworthy events happening at the W3C.

In addition, both systems incorporate a basic ontology about people, places, events, organizations, etc., which comes from the TAP Knowledge base. This ontology defines a large number of basic vocabulary terms which are widely applicable across a number of applications.

5. AUGMENTING SEARCH WITH DATA

As described earlier, we have two goals in Semantic Search. The first is to augment traditional search results with data pulled from the Semantic Web. The second is to use an understanding of the denotation of the search term to improve traditional search. In this section we describe how Semantic Search augments the results of traditional search. There are three main problems to be addressed in doing this.

Denotation: We need to determine the concept denoted by the search query, if any.

What to show: We need to determine what relevant data to pull from the Semantic Web.

Presentation: We need to appropriately format the data/triples for inclusion in the search results.

We look at each of these problems in turn. Before that, we first discuss an assumption made by Semantic Search about the data and then briefly discuss the system architecture.

5.1 Assumptions about the Data

Different sites on the Semantic Web might provide different kinds of data about an object. In general, it will be difficult, if not impossible, to control what kind of data is available about any given object. This is in stark contrast to traditional relational database systems which have fixed schemas and integrity constraints which ensure that a uniform set of information is available about any given type of object. This lack of uniformity in the data available about any particular object can be problematic for applications on the Semantic Web. Any given application typically has very specific requirements for the kinds of data it needs. On the other hand, Semantic Web applications should also be able to exploit new and interesting pieces of data that might become available on the Semantic Web. Hence, we need to balance the variations in the data availability by making sure that at a minimum, certain properties are available for all objects. In particular, we make sure that for each object, we have data about what kind of object it is and how it is usually referred to, i.e., that its *rdf:type* and *rdfs:label* is available. In addition we also make sure that a small number (preferably just one) of data sources have this core information for all of the objects that we are interested in. In some contexts, we may be able to rely on certain data sources providing certain pieces of data. In such cases, the application should be able to exploit this.

5.2 System Architecture

The Semantic Search application runs as a client of the TAP infrastructure. It runs alongside the search engine. When the search query is recieved, the search front end, in addition to sending the query to the search backend, also invokes the Semantic Search application. The Semantic Search application access the Semantic Web via the TAP client interfaces (GetData, search and reflection interfaces) to perform the three steps mentioned above.

5.3 Choosing a Denotation

The first step is to map the search term to one or more nodes of the Semantic Web. This is done by using the search interface provided by TAP. A TAP search query is issued to the one (or few) sites known to contain the *rdf:type* and *rdfs:label* information for all the objects we are interested in. This might return no matches (i.e., no candidate denotations), a single match or multiple matches. There are two ways in which a search term may map to more than one node in the Semantic Web.

Ambiguity: More than one term in the Semantic Web has the search term (or a subset of the search term) as its rdfs:label or one of the other properties indexed by the search interface. For example, in ABS, the search query "Paris" could either map to the city Paris, France or the city Paris, Texas, the music group Paris or ... In this case, we have to pick one of these as a preferred denotation. This choice can be made based on a number of different factors, including,

- The popularity of the term as measured by its frequency of occurrence in a text corpus or the availability of data on the Semantic Web. E.g., Paris, France is a preferred denotation for "Paris", compared to Paris, Texas, the music group Paris, etc.
- The user profile may be guide selection of the denotation. E.g., the phrase "Quark", as used by a Star Trek fan might be more likely to denote the Star Trek character than the subatomic particle.
- The search context can also help select the denotation. If the user has been searching for information about musicians, the query "Pink" may more likely denote the musician than the color.

Denotations other than the preferred one are also offered to the user by the system as part of the search user interface so that the user can select one of those if that is what s/he intended.

Complex Search Term: In some cases, subsets of the search term map to different nodes. For example, the query "eric miller rdf" can be broken down into "eric miller"+"rdf", the first mapping to the node corresponding to the person Eric Miller and the second mapping to the Resource Description Framework. It is possible to take this too far and map any query, including navigational queries into complex search terms. To avoid this, we restrict complex search terms to only two denotations.

If the search term does not denote anything known to the Semantic Web, then we are not able to contribute anything to the search results. Hence it is important for the Semantic Web to have knowledge of a very broad range of terms. In our experiments, ABS covers about 17% of the searches encountered on the Website of the Open Directory Project [4]. Early analysis of the service running at the W3C indicate the local knowledge base covers 85% of the most common search queries encountered on the Website.

5.4 Determining What to Show

Once we have either a single node (or a pair of nodes) corresponding to the search term, the next task is to determine what data from the Semantic Web we should incorporate as part of the search results, and in what order. As with traditional search, deciding what to show and in what order is one of the central problems of Semantic Search. This problem can be visualized in terms of the Semantic Web graph as follows. The node which is the selected denotation of the search term, provides a starting point. We will refer to this node as the *Anchor Node*. We then have to choose a subgraph around this

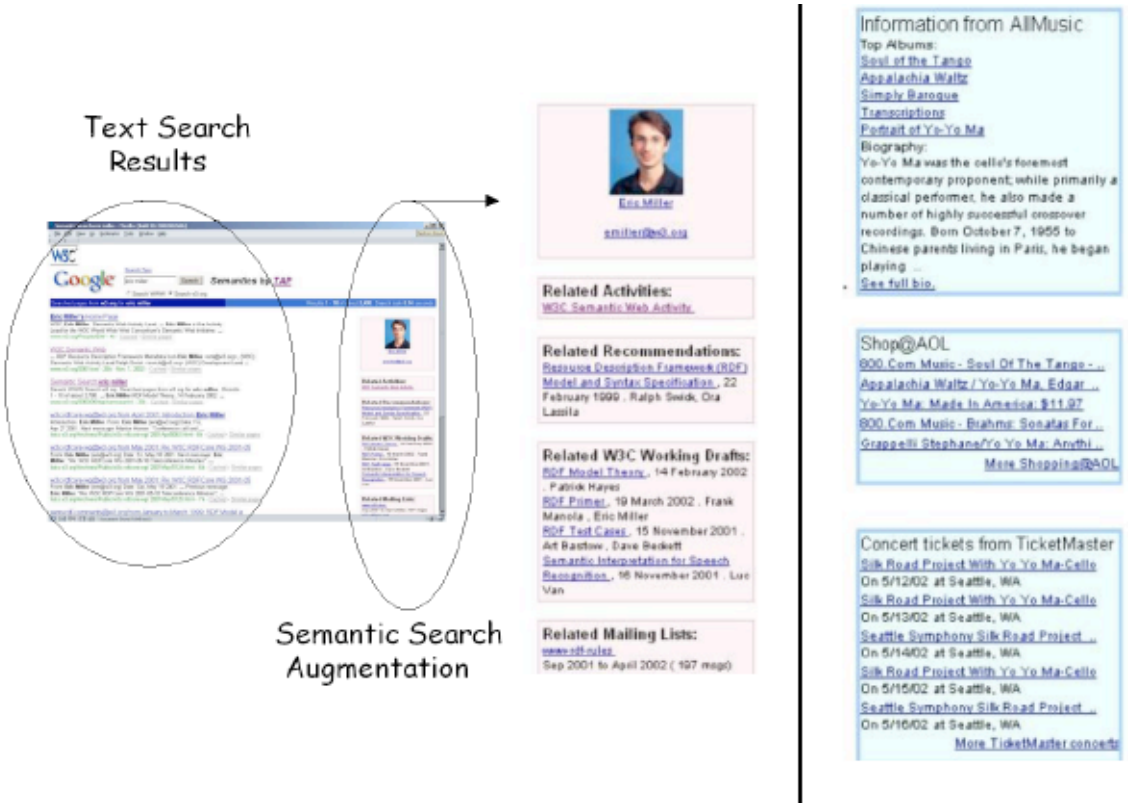

Figure 3: Text Search Results Augmentated with Data from the DataWeb. Left : Search for 'Eric Miller' on the W3C Web site showing overall page and data augmentation. Right: Data augmentation alone for search on 'Yo-Yo Ma'

node which we want to show. In the case of the search term denoting a combination of terms, we have two anchor nodes from which we have to choose the subgraph. Having chosen the subgraph, we have to decide the order in which to serialize this subgraph in the results presented to the user.

We start with a simple syntactic approach which is widely applicable but has several limitations. We then introduce various modifications and enhancements to arrive at the approach which provides adequate results.

The simple approach for selecting the subgraph, based purely on the structure of the graph, treating all property types as equally relevant, would be to walk the graph in a breadth first order, starting with the anchor node, collecting the first N triples, where N is some predefined limit. This basic approach can be improved by incorporating different kinds of heuristics in how the walk is done, so as to produce a more balanced subgraph. Some heuristics include,

- Include at most N triples with the same source and same arc label, where N is either preset or computed based on the average branching factor (i.e., the bushiness) of the graph around the anchor node.
- Include at most M triples with the same source, where M is either preset or computed based on the bushiness of the graph around the anchor node. Further, M could be a function of the distance of the node from the anchor node. This heuristic results in the inclusion of more information about nodes closer to the anchor node.

The graph collection procedure is modified for the case of pairs of nodes denoted by the search term. In this case, in addition to collecting triples in the vicinity of each of the nodes, we also locate one or more paths connecting the nodes and include the triples involved in these paths.

The intuition behind this approach is that proximity in the graph reflects mutual relevance between nodes. This approach has the advantage of not requiring any hand-coding but has the disadvantage of being very sensitive to the representational choices made by the source on the Semantic Web. This approach also has the property of being able to incorporate new pieces of information, about the anchor node and its neighbors, as they appear on the Semantic Web, without changing anything in the Semantic Search engine. This property is both a feature and a bug. It enables Semantic Search to provide richer results as the Semantic Web grows, but also makes the system more susceptible to spam and irrelevant information.

Another problem with this approach is that it ignore the search context. For example, a search for "Eric Miller" on the W3C Website should use different data from the Semantic Web compared to the same search on the Miller family Web site. There is no way of getting this effect with this approach.

A different approach is to manually specify, for each class of object we are interested in, the set of properties that should be gathered. We do a breadth-first walk of the graph, collecting values for these properties till we have gathered a certain number of triples. For example, we specify that for instances of the class W3CStaff (call the instance *O*), we collect triples and values whose target or source is *O* and whose property is one of *title*, *imageURL*, *involvedInActivity*, *hasAuthor* and *hasEmailAddress*. So, for example, if the anchor node corresponds to Eric Miller, this will give us a set of triples and also a new set of nodes (the nodes corresponding to the Semantic Web Activity and the RDF Primer) which are the targets/sources of the triples with the properties involvedInActivity and hasAuthor. For each of these nodes, based on the type of the node we collect a new set of triples. This process continues till we either run out of new triples or we have collected enough triples.

This approach has the advantage that specifying only certain properties provides a kind of filter, thereby producing more dependable results. It is also easily customizable so as to be able to factor the search context in determining which properties should be retrieved. However, it requires more work to set up, and is also not able to incorporate new kinds of information as it appears on the Semantic Web.

A hybrid approach, which has most of the benefits of both these approaches is as follows. We first apply the second approach to the extent we have specifying properties available. If not enough data has been collected, we do a more general graph walk, as per the first approach, only looking at property types not examined in the first pass.

The triples thus collected are clustered and ordered, first by the source, based on the proximity to the Anchor Node and then by the arc-label.

5.5 Formatting

The final problem is that of showing the data thus collected to the user. This is really a function of the overall search user interface, but for the sake of completeness, we describe how the Semantic Search application does this.

The display of the triples (independent of how the triples are collected) in the search results page is done by a set of templates. With each class of objects we are interested in, we associate an ordered set of templates. Each template specifies the class it applies to and the properties (of the templated object) that must be available for it to apply, and an HTML template for presenting the results. We walk through the ordered list of nodes collected, identify the template for it, generate the HTML and include them in the user interface of the search results. There may some triples left over, i.e., not covered by the chosen template. These are covered by smaller templates associated with each property type, each of which formats a single triple at a time. The templates can be encoded either in a declarative templating language we have developed or in a scripting language such as Perl.

Figure 3 show the results of search augmentations for search *Eric Miller* in W3C Semantic Search and for *Yo-Yo Ma* in ABS.

5.6 Emperical Evaluation

We are in the process of empirically evaluating the usefulness of augmenting search with data from the Semantic Web. This test is being done with W3C Semantic Search. A randomly chosen anonymous set of people doing search on the W3C website are presented with the W3C Semantic Search instead of the regular search results. On each of these pages, each of the regular search results and links in the data augmenting the search is sent through a redirector which records the search query, the link and which section of the page the link was on. The fraction of times the user clicked on a link in the data augmentation (as opposed to one in the regular search results) tells us how useful the user found the data augmentation.

6. SEMANTICS FOR TEXT SEARCH

In the last section we described how Semantic Search augments the results of traditional search with data from the Semantic Web. We would also like to be able to use the Semantic Web to improve the results of the text search itself. Intuitively, the text search should be able to exploit an understanding more about what the user is trying to find information about.

Work in Latent Semantic Indexing [15] and related areas has explored the use of semantics for information retrieval. However, much of that work has focussed on generating the semantic structures from text. With the Semantic Web, we are already given a large-scale, albeit distributed, *explicit* semantic structure, constructed independently from the text being searched.

There will likely be many different ways in which data from the Semantic Web will get used to filter and rank text search results. In this section, we describe one particular problem we are trying to solve.

Google has about 136,000 results for a search on "Yo-Yo Ma". Manual analysis of the first 500 of these results shows that all 500 of them refer to the same person, i.e., the famous cellist Yo-Yo Ma. On the other hand, the search "Eric Miller" yields 1,400,000 results, the first 20 of which refer to at least 16 different Eric Millers. Similarly, the search "Matrix" yields 4,380,000 results and included in the first 20 are references to a movie, the mathematical concept, four different companies, a data center, a monastery, a display device, a hair care product and an on-line community. The user is likely searching for information about one particular Eric Miller or one of the denotations of the word "Matrix". Unfortunately, there is no easy easy way of communicating this to the system.

Our goal is to enable a search engine to understand that different occurrences of the same string denote different things and to further filter and rank the results to show documents referring to the chosen denotation. Our initial focus is on search queries denoting people.

We first need to provide the user with a simple, unobtrusive mechanism for identifying the right denotation. In some cases (e.g., the word "jaguar"), there are a small number of primary denotations (the animal, the car and the sports team) and we can pick one and list the others (at the top of the search results) to let the user pick one of the others as the intended denotation. In many other cases (such as Matrix or Eric Miller), where there are many thousands of potential denotations, the approach of listing all the possible denotations separately from search results does not work. So, we modify the presentation of the search results so that each search result has an additional link next to it with which the user can tell the search engine *this is the intended denotation*. For example, applied to the Google search interface, the first result on the search for "Eric Miller" could look like the following:

Eric Miller's Home Page
W3C, Eric Miller. Semantic Web Activity Lead. ...
Lead for the W3C World Wide Web Consortium' ...
www.w3.org/EM/ - 4k - Cached - Similar pages - **This Eric Miller**

Once the user has picked the right denotation, we now face the bigger problem of determining the likelihood of a document denoting our chosen denotation.

We provide a brief summary of the approach we are taking. We are using a knowledge based approach in which we identify and encode a number of different heuristics, each of which might apply in different cases. We list three sample heurisitcs here:

- Just knowing that the user is searching for information about a person (as opposed to documents containing the two strings "Eric" and "Miller") can help avoid several mistakes. It can help eliminate documents in which both these words occur, but in completely different parts of the document. For example, it can enable the search engine to realize that a document with the sentence "Rob is Cool" or "Eric likes Miller beer" is likely not as accurate a result as a document with the sentence "Someone with an email address of (robm@ksl.stanford... or em@w3.org) said that ...".
- The type of the person (denoted by the search term) drives expectations about various categories of information that might be available about it, which in turn can help sort the search results. For example, if the person is a musician, we can expect to find pages about albums, about his concert schedule, fan pages, etc. On the other hand, if the object is a professor, we can expect to find pages related to his papers, course offerings and research. We don't expect to find research papers by musicians or concert schedules for professors. This heuristic can be used when the Semantic Web knows about the intended denotation.
- It might be a while before the Semantic Web knows about all the Eric Millers referenced on the Web. Even in this case, where the Semantic Web does not yet know about the intended denotation, knowledge about people in general can be used. For example the co-occurrence of unique string valued properties of the chosen person (such as his/her email address), can help the system decide whether two documents are referring to the same Eric Miller.

Given that most searches on the Web are about a relatively small number of categories, and that each category only needs a few heuristics, we believe that this approach will greatly help users identify which object they are searching for information about and get documents with a high likelihood of referring to that object ranked higher in the search results.

7. CONCLUSIONS AND FUTURE WORK

The widespread availability of machine understandable information has the potential to deeply impact many important Web applications, including search. In particular, we believe that research oriented search queries can significantly exploit the emerging Semantic Web. In this paper, we showed two mechanisms by which this might happen, with an emphasis on augmenting search results from the Semantic Web. Our future work is focused on helping the text search component of such a hybrid system exploit a deeper understanding of the search term's denotation.

8. ACKNOWLEDGEMENTS

The material presented here is the work of many people over a long period of time. The original TAP infrastructure was built as a part of Alpiri Inc. We thank all the people who made that possible, in particular Arvind Sundarajan and Kate Joly. We would like to thank Ed Feigenbaum, Richard Fikes, Deborah McGuiness and Shiela McIlraith of the Knowledge System Lab at Stanford. We would

also like to thank Dan Brickley. The first author would like to thank IBM Almaden for its help and support.

9. REFERENCES

[1] The apache http server. `http://www.apache.org/`.

[2] Daml query language. `http://www.daml.org/dql/`.

[3] Darpa agent markup language. `http://www.daml.org/`.

[4] The open directory project. `http://www.dmoz.org/`.

[5] Rdf site summary. `http://www.purl.org/rss/1.0/`.

[6] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery 1.0: An XML query language. http://www.w3.org/TR/xquery/, 30 April 2002. W3C working draft.

[7] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winder. Simple Object Access Protocol. `http://www.w3.org/TR/SOAP/`, May 2000.

[8] D. Brickley and R. V. Guha. Rdf schema. `http://www.w3.org/TR/rdf-schema/`.

[9] Google. `http://www.google.com`.

[10] J. Heflin and J. Hendler. Searching the web with shoe. In *AAAI-2000 Workshop on AI for Web Search*.

[11] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. Rql: a declarative query language for rdf. In *WWW*, pages 592–603, 2002.

[12] B. McBride. Jena: Implementing the rdf model and syntax specification. http://www-uk.hpl.hp.com/people/bwm/papers/20001221-paper/, 2001. Hewlett Packard Laboratories.

[13] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(3):54, 1997.

[14] L. Miller, A. Seaborne, and A. Reggiori. Three implementations of squishql, a simple rdf query language. In *International Semantic Web Conference*, pages 423–435, 2002.

[15] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. pages 159–168, 1998.

[16] R.Guha and R. McCool. The tap knowledge base. `http://tap.stanford.edu/`.

[17] R.Guha and R. McCool. Tap: Towards a web of data. `http://tap.stanford.edu/`.

[18] R. Siebes and F. van Harmelen. Ranking agent statements for building evolving ontologies. In P. Bouquet, editor, *Workshop on Meaning Negotiation, in conjunction with the Eighteenth National Conference on Artificial Intelligence*, July 2002.