A Portrait of the Semantic Web in Action

Jeff Heflin and James Hendler
University of Maryland

The World Wide Web's phenomenal growth rate is making it increasingly difficult to locate, organize, and integrate the available information. To cope with this enormous quantity of data, we need to hand off portions of these tasks to machines. However, since natural language processing is still an unsolved problem, machines cannot understand the web pages to the extent required to perform the desired tasks. An alternative is to change the Web to make it more understandable by machines, thereby creating a Semantic Web. Many researchers believe that the key to building this new web lies in the development of semantically-enriched languages. Early languages, such as RDF, SHOE, and Ontobroker, have led to more recent efforts, such as the DARPA Agent Markup Language (DAML). It is argued that languages such as these will revolutionize the Web, but if so, how will the new Web work?

In this article, we will put a Semantic Web language through its paces and try to answer questions about how it can be used in practice. How are the semantic descriptions generated? How are these descriptions discovered by agents? How can information from different sites be integrated? How can the Semantic Web be queried? We will present a system that addresses these questions, and describe a suite of tools to help users in their various interactions with the Semantic Web. Throughout this article, we'll motivate the design of our system by a specific application: semantic markup in the computer science domain.

Overview of Semantic Web Languages

Unlike ordinary XML documents, where the meaning of things is often implied by a name or prose description, a Semantic Web language must describe meaning in a machine readable way. Therefore, the language needs not only the ability to specify a vocabulary, but also the ability to formally define the vocabulary so that it can be used in automated reasoning. As such, Semantic Web languages are greatly influenced by the subfield of AI known as knowledge representation (KR). However, in order to meet the needs of the Web, they must also differ from traditional KR languages. The most obvious difference is syntactical: the syntaxes of Semantic Web languages are based on existing standards such as HTML or XML, so that integration with other Web technologies is possible. Other differences depend on the very nature of the Web. Since the Web is decentralized, the language must allow for the definition of diverse, and potentially conflicting, vocabularies. To handle the rapid evolution of the Web, the language must allow for the vocabularies to evolve as human understanding of their use improves. Finally, the immense size of the Web requires that scalability play a role in any solution.

A Semantic Web vocabulary can be formally specified using an ontology or schema. Ontologies are also typically sharable (so that different users can agree to use the same definitions) and extensible (so that users can agree on some common set of definitions but add terms and definitions as necessary). It is expected that ontology hierarchies will develop, with top-level abstract ontologies at the root and domain-specific ontologies at the leaves; in this way automatic interoperability between a pair of ontologies exists to the degree that they share a common ancestor. The potential richness of an ontology's definitions is determined by the expressivity of the language. Most languages allow ontologies to define class taxonomies, so that it is possible to say, for example, that a Car is a type of Vehicle. They also allow properties to be defined for each class, and/or relationships to be defined between multiple classes. Some

languages may also allow more complex definitions to be formed, using axioms from some form of logic.

There are major differences between the leading Semantic Web languages. The Resource Description Framework (RDF) with RDF Schema[1] is the least expressive of them. It is based on a semantic network model, with special links for defining category and property taxonomies, and links for applying domain and range constraints to properties. SHOE (Simple HTML Ontology Extensions)[2] is based on a frame system, but also allows Horn clause axioms, which can be used to define things not possible in RDF. More so than its peers, SHOE focuses on dealing with the problems of a dynamic, distributed environment.[3] OIL (the Ontology Inference Language)[4] is based on a frame system augmented with description logic, which allows different kinds of definitions to be expressed. The DARPA Agent Markup Language (DAML) is a language currently under development with the intent to combine the best features of RDF, SHOE, and OIL. The Ontobroker project[5] uses a language that differs from the others in that its ontologies are defined externally.

Although ontologies are crucial to making a Semantic Web language work, they merely serve to standardize and provide interpretations for the content of web pages. To make this content machine understandable, the web pages must contain semantic markup, that is, descriptions that use the terminology defined in one or more ontologies. The semantic markup may state that a particular entity is a member of a class, that an entity has a particular property, or that two entities have some relationship between them. By committing to an ontology, the semantic markup sanctions inferences based on the ontology definitions, so that it is possible to conclude things that were implied by the markup.

Producing Semantic Markup

The task of describing a set of web pages using a Semantic Web language can be challenging. The first step is to consider the domain of the pages and choose an appropriate ontology. As Semantic Web languages evolve, it is likely that huge ontology libraries will become available, and numerous search mechanisms will be provided to help users find relevant ontologies. In the meantime, some of the common languages provide libraries of starter ontologies. The design of ontologies can be a difficult process, and is covered by the discipline of knowledge engineering, which is outside the scope of this article.

In order to be specific, our running example will use the SHOE language, which has served as a testbed for Semantic Web ideas over the past five years, although technically the discussion could apply to any Semantic Web language. SHOE has a computer science department ontology that includes classes such as Student, Faculty, Course, Department, Publication, and Research, and relations such as publicationAuthor, member, emailAddress, and advisor. The scope of this ontology makes it relevant to faculty and student homepages, department web pages, project web pages, and publication indices. A number of methods can be used to produce SHOE markup for these pages.

Authoring Tools

As with HTML, semantic markup can be added to a page using a simple text editor. However, unlike HTML processors, Semantic Web processors are not very forgiving, and errors can result in large portions of the annotations being ignored. One solution is to provide authoring tools that allow markup to be created by making selections and filling in forms; for the SHOE project, we have developed the Knowledge Annotator to perform this function. This tool has an interface that displays instances, ontologies, and claims (see Figure 1). Users can add, edit or remove any of these objects. When creating a new object, users are prompted for the necessary information. In the case of claims, a user can choose the source ontology from a list, and then choose categories

or relations defined in that ontology from another list. The available relations is automatically filtered based upon whether the instances entered can fill the argument positions. A variety of methods can be used to view the knowledge in the document. These include a view of the source HTML, a logical notation view, and a view that organizes claims by subject and describes them using simple English. In addition to prompting the user for inputs, the tool performs error checking to ensure correctness and converts the inputs into legal SHOE syntax. For these reasons, only a rudimentary understanding of SHOE is necessary to markup web pages. If contemporary web authoring tools are enhanced to include semantic markup authoring capabilities, then adding semantic markup may become a natural part of the web page development process.

Various members of our research group provided markup for their home pages and as well as pages of the group's website. Most used the Knowledge Annotator, but some preferred to use a text editor. Although this produced detailed markup for a set of pages, the set is too small to be of use for anything but controlled demos.

Generating Markup on a Large Scale

If semantic markup is to be really useful, it needs to be ubiquitous, but it is tedious to use an authoring tool to generate large amounts of markup. In fact, detractors of the Semantic Web language approach often cite the difficulty in producing markup as the main reason why it will never work. Fortunately, there are many approaches for automatically generating semantic markup.

Some web pages have regular structure, with labeled fields, lists, and tables. Often, these structures can be mapped to an ontology, and a program can be written to translate portions of the web page into the semantic markup language. We developed Running SHOE (shown in Figure 2), a tool that helps users specify how to extract SHOE markup from these kinds of web pages. The user selects a page to markup and creates a wrapper for it by specifying a series of delimiters that describe how to extract interesting information. These delimiters indicate the start of a list (so that header information can be skipped), the end of a list (so that trailing information can be ignored), the start of a record, the end of a record, and for each field of interest, a pair of start and end delimiters.

A fundamental problem in distributed systems is knowing when markup authored by different people describes the same entity. If we are to integrate descriptions about this entity, then a common identifier (or key) must be used when referring to it. A Uniform Resource Locator (URL) can often serve as this key because it identifies exactly one web page, which is owned by a single person or organization. The regular pages that work best with Running SHOE tend to have lists of things, and each item in the list typically contains a hyperlink to the homepage of the thing in question. However, these hyperlinks often use relative URLs, which are not unique. To handle this problem, the user can specify that a particular field is a URL, so that when data is extracted, all relative URLs are expanded using the page's URL as a base.

After the user has specified the delimiters, the tool can display a table with a row for each record and a column for each field. Irregularities in a page's HTML can often cause fields or records to be extracted improperly, and this table allows the user to verify the results before proceeding. The next step is to convert the table into SHOE markup. In the top right panel, the user can specify ontology information and a series of templates for SHOE classification and relation declarations. For each classification or relation argument, the user can specify a literal value or reference a field. At the user's command, the tool can then iterate through these templates and the table of records to create a series of SHOE statements. Using this tool, a trained user can extract substantial markup from a web page in minutes. Furthermore, since

Running SHOE allows these templates to be saved and retrieved, it is easy to regenerate new SHOE markup if the content of the page changes.

Computer science department websites often have faculty lists, project lists, course lists, and user lists that are of the correct format for Running SHOE. Each item in the list contains an <A> tag that provides the URL of the item's homepage, and the content of this element is often the name of the entity being pointed to, providing us with a value for the "name" relation. Other properties of the instance often follow and are delimited by punctuation, emphasis, or special spacing. A single user was able to create SHOE markup about the faculty, courses, and projects of fifteen major computer science departments in a day.

Although there are many pages for which Running SHOE is useful, there are other important resources from which it cannot extract information. An example of such a site is CiteSeer (http://citeseer.nj.nec.com/cs), an index of online CS publications that we wanted to integrate with our department web pages. Interaction with CiteSeer involves issuing a query to one page, viewing a results page, and then selecting a result to get a page about a particular publication. It is this multi-step process that prevents Running SHOE from extracting markup from this website.

To extract SHOE from CiteSeer, we built another tool called Publication SHOE Maker (PSM). PSM issues a query to get publications likely to be from a particular institution, and retrieves some fixed number of publication pages from the results. The publication pages contain the publication's title, authors, year, links to online copies, and occasionally additional BibTex information. The layout of each publication page is very similar, so the values of the desired fields can be easily extracted.

An important issue is how to link the author information with the faculty instances extracted from the department web pages. Fortunately, CiteSeer includes homepage information, generated by HomePageSearch (http://hpsearch.uni-trier.de/), for each author. By using these URLs, as opposed to the author's names, links to the appropriate instances can be established.

Running SHOE and PSM are only two examples of tools that can be used to generate large quantities of markup. Other extraction tools might include machine learning[6-7] or natural language processing techniques. As XML becomes ubiquitous on the Web, it will become easier to generate wrappers, and stylesheets can be used to transform a simple XML vocabulary into a semantically-enriched one. If the author of a web page is willing to provide semantic markup, then the process can be even easier. For example, much of the Web's data is kept in databases, and scripts are used to produce web pages from the content of these databases. Since databases are structured resources, it is possible to determine the semantics of each structure and map it to an ontology. Once a database has been mapped to an ontology, it is easy to modify the script that produces web pages to include semantic markup as well.

Integrating Resources

After a number of diverse web sites have been described with semantic markup, the next problem is determining how to integrate this information. Information integration systems, such as Ariadne[8], can be extremely useful when developing an application that combines data from a finite number of predetermined sources, but are less helpful when there is a need to integrate information "on the fly." We will examine a solution that mirrors the operation of contemporary search engines by crawling the Web and storing the information in a central repository.

Exposé is a web-crawler that searches for web pages with SHOE markup and interns the knowledge. A web-crawler essentially performs a graph traversal where the nodes are web pages and the arcs are the hypertext links between them. When Exposé discovers a new URL, it assigns the page a cost and uses this cost to determine where it will be placed in a queue of URLs to be visited. In this way, the cost function determines the order of the traversal. We assume that

SHOE pages will tend to be localized and interconnected. For this reason, we currently use a cost function which increases with distance from the start node, where paths through non-SHOE pages are more expensive than those through SHOE pages, and paths that stay within the same directory on the same server are cheaper than those that do not. Exposé parse each web page, and if a page references an ontology that Exposé is unfamiliar with, it loads the ontology as well. In order to update its list of pages to visit, it identifies all of the hypertext links, category instances, and relation arguments within the page, and evaluates each new URL as above. Finally, the agent stores SHOE category and relation statements, as well as any new ontology information, in a knowledge base (KB).

This knowledge base will determine the query capabilities of the system, and thus an appropriate knowledge representation system must be chosen. Our SHOE tools all use a generic API for interaction with the knowledge base, allowing us to easily use different backends. We have currently implemented versions of this API for Parka, a high performance frame system;[9] XSB, a deductive database;[10] and OKBC compliant knowledge bases.[11] By changing the backend knowledge representation system, we get varying tradeoffs between query response time and the degree to which inference is used in computing the answers. For example, Parka has been shown to answer recognition queries on large KBs (containing millions of assertions) in seconds, and when used on parallel machines, it provides even better performance. However, Parka's only inferential capabilities are class membership and inheritance, thus it can not make use of the extra Horn clause rules allowed by SHOE. On the other hand, XSB can reason with these rules, but is not as efficient as Parka. Alternatively, the KB could be a relational or object database; this would provide the greatest scalability and best query response times, but would sacrifice the ability to do inference. Clearly, the choice of the backend system depends on the expected query needs of the application.

We let Exposé crawl the various computer science web pages described earlier, and it was able to gather approximately 40,000 assertions. The crawler stored these assertions in both Parka and XSB KBs. Note that technically we did not need a web-crawler for our example, because we knew the locations of all of the relevant pages a priori. However, in an ideal Semantic Web situation, the markup is the product of many individuals working independently and could not be easily located without a crawler.

Querying the Semantic Web

Both general purpose and domain specific query tools can access the SHOE knowledge after it has been loaded into the KB. The SHOE Search tool (see Figure 3) is a general purpose tool that gives users a new way to browse the web by allowing them to submit structured queries and open documents by clicking on the URLs in the results. The user first chooses an ontology against which the query should be issued; this essentially establishes a context for the query. The user then chooses the class of the desired object from a hierarchical list and is presented with a list of all properties that could apply to that object. After typing in desired values for one or more of these properties, the user issues a query and receives a set of results in a tabular form. If the user double-clicks on a binding that is a URL, then the corresponding web page will open in a new window of the user's web browser. In this way, the user can browse the Semantic Web.

It may be the case that all of the relevant web pages are not described by SHOE markup. In such cases, the standard query method of SHOE Search will not be able to return an answer, or may only return partial answers. Therefore, we also have a **Web Search** feature that translates the user's query into a similar search engine query and submits it to any one of a number of popular search engines. Using SHOE Search in this way has two advantages over using the search engines directly. First, by prompting the user for values of properties it increases the chance that the user will provide distinguishing information for the desired results. Second, by

automatically creating the query it can take advantage of helpful features that are often overlooked by users such as quoting phrases or using the plus sign to indicate a mandatory term. Currently, we build a query string that consists of a quoted short name for the selected category and, for each property value specified by the user, a short phrase describing the property followed by the user's value, which is quoted and preceded by a plus sign to indicate that it is a mandatory phrase.

With SHOE Search, a user can ask many queries pertinent to our computer science domain. Figure 3 displays a sample query to locate faculty members from the University of Washington and their publications. The CS ontology serves as a unifying framework for integrating information from the university's faculty page with publication information extracted from CiteSeer. Furthermore, the ontology allows the query system to recognize that anyone who is declared to be a Professor is also Faculty.

Sample queries to the knowledge base exposed one problem with the system: sometimes information from a department web page and CiteSeer was not integrated as expected. It turns out that these sites occasionally use different URLs to refer to the same person. This is a fundamental problem with using URLs as keys in a Semantic Web system: multiple URLs can refer to the same web page, due to things like multiple host names for a given IP address, default pages for a directory URL, host-dependent shortcuts such as "~" for the users directory, and symbolic links within the host. Additionally, some individuals may have multiple URLs that make equally valid keys for them, such as the URLs of a professional homepage and a personal homepage. These problems would be partially alleviated if the language included the ability to specify identifier equivalence: a feature absent from SHOE, but present in DAML.

We have created a search engine called Semantic Search (see Figure 4) that is based on the technologies described here. Semantic Search uses the SHOE Search tool as a query interface and provides utilities for authors. There are links to an ontology library, the Knowledge Annotator, an online SHOE validation form, and a form for submitting new pages to the repository. The interested reader is encouraged to add markup up to their own web pages and submit them. Semantic Search is available at http://www.cs.umd.edu/projects/plus/SHOE/search/.

Conclusion

We have described a simple architecture for a Semantic Web system that parallels the way contemporary web tools and search engines work. As depicted in Figure 5, various tools are used to add markup to web pages, and a web-crawler discovers the information and stores it in a repository, which can then be queried by other tools. In particular, we have shown that not all markup needs to be produced by hand; in many cases, simple extraction tools can generate accurate markup with minimal human effort. Although the tools that comprise this architecture were designed for use with the SHOE language, similar tools can be created for other Semantic Web languages. Since any number of tools can produce and process the semantic markup on a web page, other architectures are also possible. For example, an agent could be developed that queries pages directly, as opposed to issuing queries to a repository constructed by a web-crawler.

It is clear that if the vision of the Semantic Web is achieved, it will become much easier to locate useful information on the Internet, and the integration of diverse resources will be simplified. Obviously there are still obstacles to overcome: we need better schemes for ensuring interoperability between independently developed ontologies and approaches for determining who and what to trust. Still, we believe the first step towards the Semantic Web vision is the design of languages that allow semantic markup to be expressed. The next step is to build systems and tools like those described in this article, so that users can easily provide and receive

information on the Semantic Web. When it becomes as easy to add semantics to web pages as it is to author contemporary web pages, then the next Internet revolution will arise.

Acknowledgments

This work was supported by the Air Force Research Laboratory under grant F306029910013.

References

1. O. Lassila, "Web Metadata: A Matter of Semantics," *IEEE Internet Computing*, Vol. 2, No. 4, July 1998, pp. 30-37.
2. S. Luke et al., "Ontology-based Web Agents," *Proc. First Int'l Conf. Autonomous Agents*, ACM Press, New York, 1997.
3. J. Heflin and J. Hendler, "Dynamic Ontologies on the Web," *Proc. 17th Nat'l Conf. AI (AAAI-2000)*, MIT/AAAI Press, Menlo Park, CA, 2000, pp. 443-449.
4. S. Decker et al., "Knowledge Representation on the Web," *Proc. 2000 Int'l Workshop on Description Logics (DL2000)*, 2000.
5. D. Fensel et al., "Ontobroker: How to Enable Intelligent Access to the WWW," *AAAI-98 Workshop on AI and Information Integration*, AAAI Press, Menlo Park, CA, 1998, pp. 36-42.
6. D. Freitag, " Information Extraction from HTML: Application of a General Machine Learning Approach," *Proc. 15th Nat'l Conf. AI (AAAI-98)*, MIT/AAAI Press, Menlo Park, CA, 1998, pp. 517-523.
7. N. Kushmerick, D. Weld, and R. Doorenbos, "Wrapper Induction for Information Extraction," *Proc. 15th Int'l Joint Conf. AI*, Morgan Kaufmann, San Francisco, 1997, pp. 729-735.
8. C. Knoblock et al., "Modeling Web Sources for Information Integration," *Proc. 15th Nat'l Conf. AI (AAAI-98)*, MIT/AAAI Press, Menlo Park, CA, 1998, pp. 211-218.
9. K. Stoffel, M. Taylor, and J. Hendler, "Efficient Management of Very Large Ontologies," *Proc. 14th Nat'l Conf. AI (AAAI-97)*, MIT/AAAI Press, Menlo Park, CA, 1997.
10. K. Sagonas, T. Swift, and D. Warren, "XSB as an Efficient Deductive Database Engine," *Proc. 1994 ACM SIGMOD Int'l Conf. On Management of Data (SIGMOD'94)*, 1994, pp. 442-453.
11. V. Chaudhri et al., "OKBC: A Programatic Foundation for Knowledge Base Interoperability," *Proc. 15th Nat'l Conf. AI (AAAI-98)*, MIT/AAAI Press, Menlo Park, CA, 1998, pp. 600-607.

Biography

Jeff Heflin is a Ph.D. candidate at the University of Maryland. He received his B.S. in computer science from the College of William and Mary in 1992 and his M.S. in computer science from the University of Maryland in 1999. He has worked in the computer consulting industry for four years as a data modeler, database designer, and database administrator. He is currently a member of the Joint US/EU Committee on Agent Markup Languages. His research interests include semantic web languages, ontologies, internet agents, and knowledge representation.

James Hendler is currently working at the Defense Advanced Research Projects Agency (DARPA) as a program manager. However, the opinions expressed in this paper are his and Mr. Heflin's own, and not do not necessarily reflect the opinions of DARPA, the Department of Defense, or any other government agency. At the University of Maryland, he is a professor and head of both the Autonomous Mobile Robotics Laboratory and the Advanced Information

Technology Laboratory. He has joint appointments in the Department of Computer Science, the Institute for Advanced Computer Studies and the Institute for Systems Research, and he is also an affiliate of the Electrical Engineering Department. He is the author of the book "Integrating Marker-Passing and Problem Solving: An activation spreading approach to improved choice in planning" and is the editor of "Expert Systems: The User Interface," "Readings in Planning" (with J. Allen and A. Tate), and "Massively Parallel AI" (with H. Kitano). He has authored over 100 technical papers in artificial intelligence, robotics, intelligent agents and high performance computing. Hendler was the recipient of a 1995 Fulbright Foundation Fellowship, is a member of the US Air Force Science Advisory Board, and is a Fellow of the American Association for Artificial Intelligence.

Contact the authors at University of Maryland, Department of Computer Science, College Park, MD 20742, USA; heflin@cs.umd.edu and hendler@cs.umd.edu.

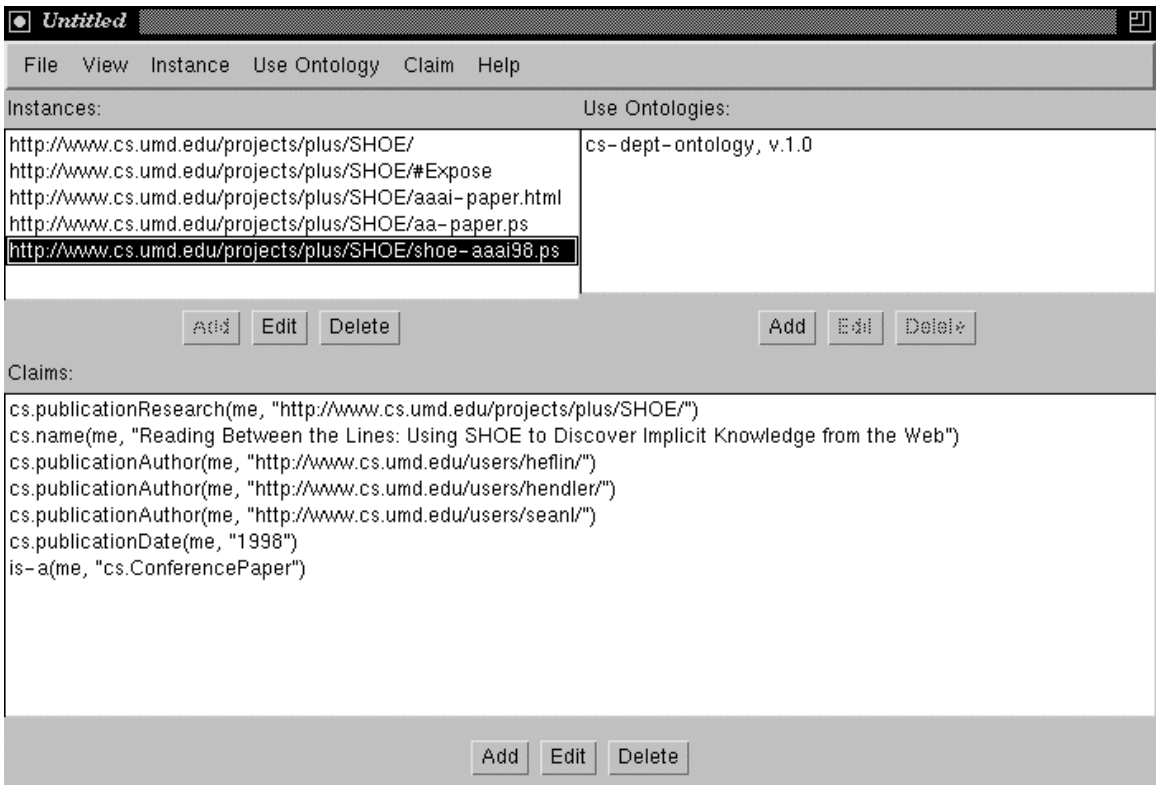

Figure 1. The Knowledge Annotator

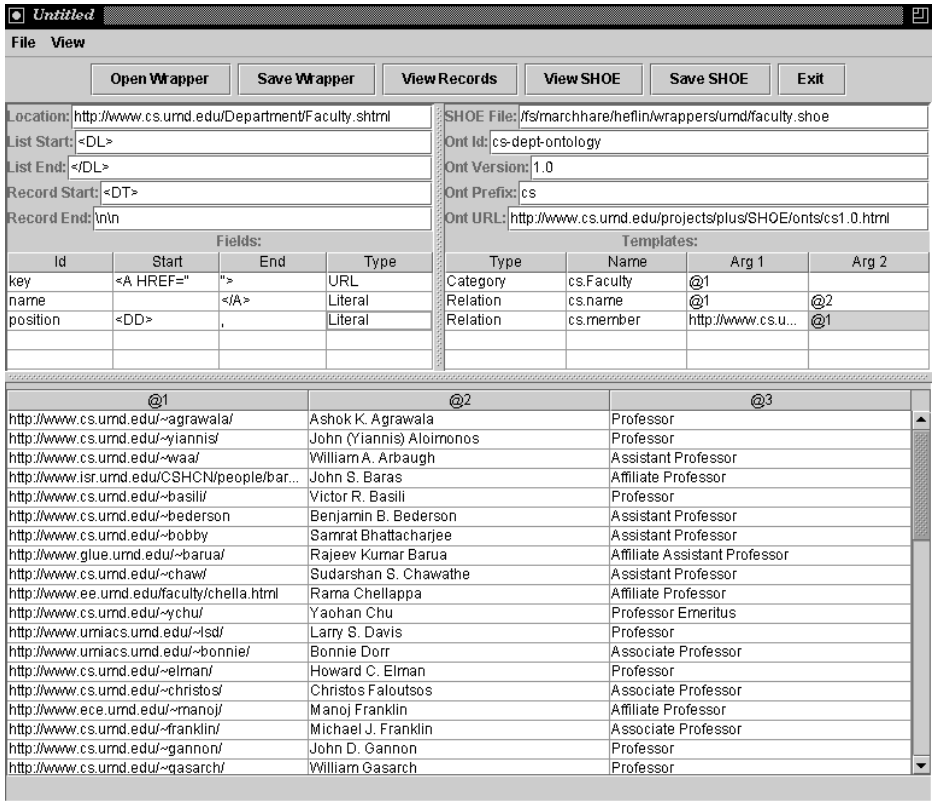

Figure 2. Running SHOE

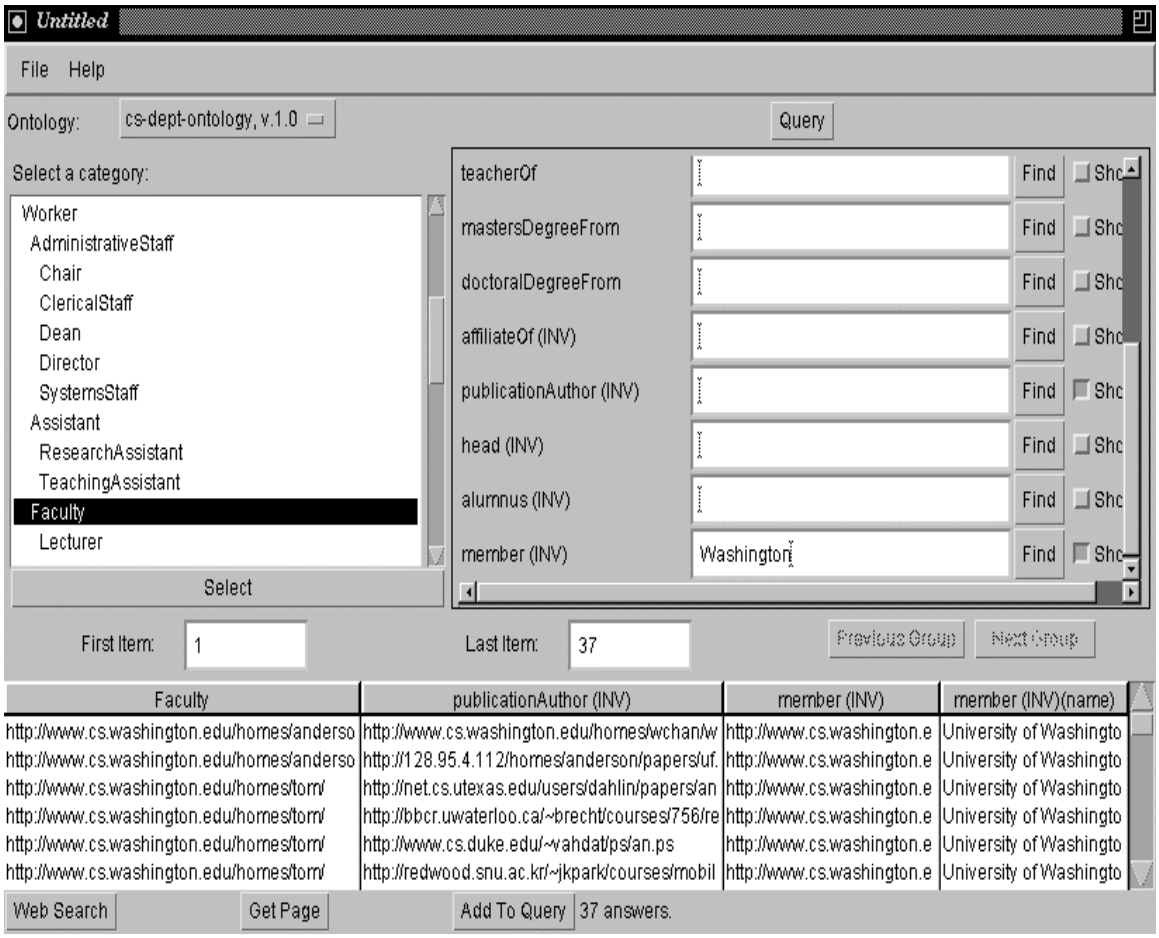

Figure 3. SHOE Search

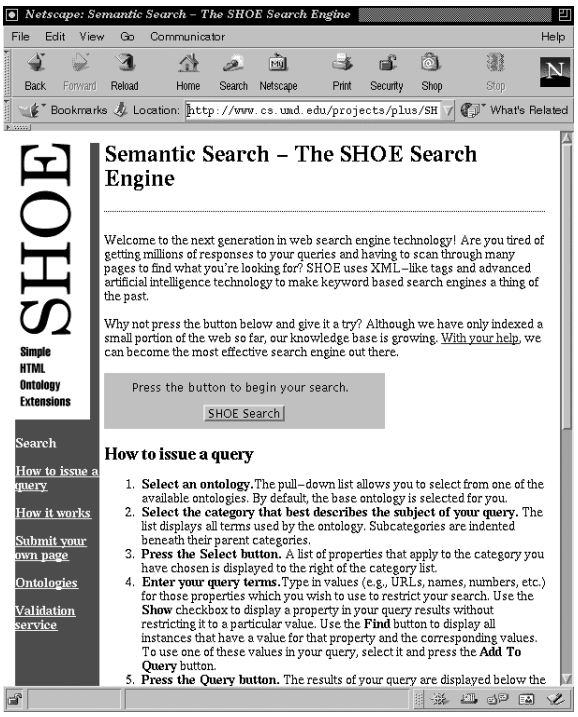

Figure 4. Semantic Search Main Page

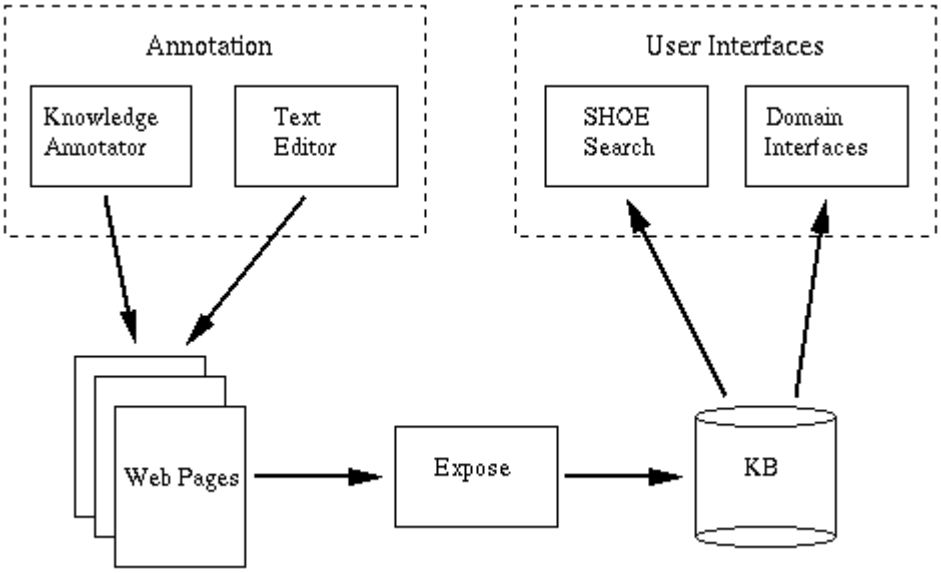

Figure 5. A Simple Semantic Web System