A Portrait of the Semantic Web in Action

Jeff Heflin and James Hendler, *University of Maryland*

The Web's phenomenal growth rate makes it increasingly difficult to locate, organize, and integrate the available information. To cope with the enormous quantity of data, we need to hand off portions of these tasks to machines. However, because natural-language processing is still an unsolved problem, machines cannot understand the Web pages to the extent required to perform the desired tasks.

Without semantically enriched content, the Web cannot reach its full potential. The authors discuss tools and techniques for generating and processing such content, thus setting a foundation upon which to build the Semantic Web.

An alternative is to change the Web to make it more understandable by machines, thereby creating a Semantic Web. Many researchers believe the key to building this new Web lies in the development of semantically enriched languages. Early languages, such as the resource description framework,[1] Simple HTML Ontology Extensions (SHOE),[2] and Ontobroker,[3] have led to more recent efforts, such as the Defense Advanced Research Projects Agency's Agent Markup Language (DAML). Some say that languages such as these will revolutionize the Web. If so, how will the new Web work?

In this article, we put a Semantic Web language through its paces and try to answer questions about how people can use it, such as:

- How do authors generate semantic descriptions?
- How do agents discover these descriptions?
- How can agents integrate information from different sites?
- How can users query the Semantic Web?

We present a system that addresses these questions and describe tools that help users interact with the Semantic Web. We motivate the design of our system with a specific application: semantic markup in the computer science domain.

Producing semantic markup

Describing a set of Web pages using a Semantic Web language can be challenging. (For an overview of Semantic Web languages, see the related sidebar.) The first step is to consider the pages' domain and choose an appropriate ontology. As Semantic Web languages evolve, knowledge engineers will likely provide huge ontology libraries, as well as numerous search mechanisms to help users find relevant ontologies. Meanwhile, some of the common languages provide starter ontology libraries. (Knowledge engineering, which covers the difficult process of designing ontologies, is outside this article's scope.)

Our running example uses 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. This ontology's scope makes it relevant to faculty and student homepages, department Web pages, research project Web pages, and publication indices. Authors can use a number of methods to produce SHOE markup for these pages.

Authoring tools

As with HTML, authors can use a text editor to

1094-7167/01/$10.00 © 2001 IEEE

add semantic markup to a page. However, unlike HTML processors, Semantic Web processors are not very forgiving, and errors can result in the processors ignoring large portions of the annotations. One solution is to provide authoring tools that let authors create markup by making selections and filling in forms. For the SHOE project, we developed the Knowledge Annotator (see Figure 1) to perform this function.

In SHOE, a document references a set of ontologies that provide the vocabulary used to describe entities (called *instances*). Each assertion about an instance is called a *claim*, to denote that it may not necessarily be true. The Knowledge Annotator has an interface that displays instances, ontologies, and claims, and a user can add, edit, or remove any of these objects. When creating a new object, the Knowledge Annotator prompts the user for the necessary information. In the case of claims, the 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 are automatically filtered based on whether the instances entered can fill the argument positions.

Users have access to various methods for viewing 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 mark up Web pages. If developers enhance contemporary Web authoring tools with semantic markup authoring capabilities, adding semantic markup could become a regular activity in the Web-page design process.

Members of our research group provided markup for their homepages and those of the

Overview of Semantic Web languages

Unlike Extensible Markup Language (XML), which uses a name or prose description to imply meaning in documents, 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 to formally define the vocabulary so that it will work in automated reasoning. As such, the subfield of AI known as knowledge representation greatly influences Semantic Web languages.

However, to meet the needs of the Web, Semantic Web languages must also differ from traditional KR languages. The most obvious difference is syntactical: Language designers base Semantic Web syntaxes on existing standards such as Hypertext Markup Language (HTML) or XML so that integration with other Web technologies is possible. Other differences depend on the nature of the Web.

Because the Web is decentralized, the language must allow for the definition of diverse—and potentially conflicting—vocabularies. To handle the Web's rapid evolution, the language must let the vocabularies evolve as human understanding of their use improves. Finally, the Web's size requires that scalability play a role in any solution.

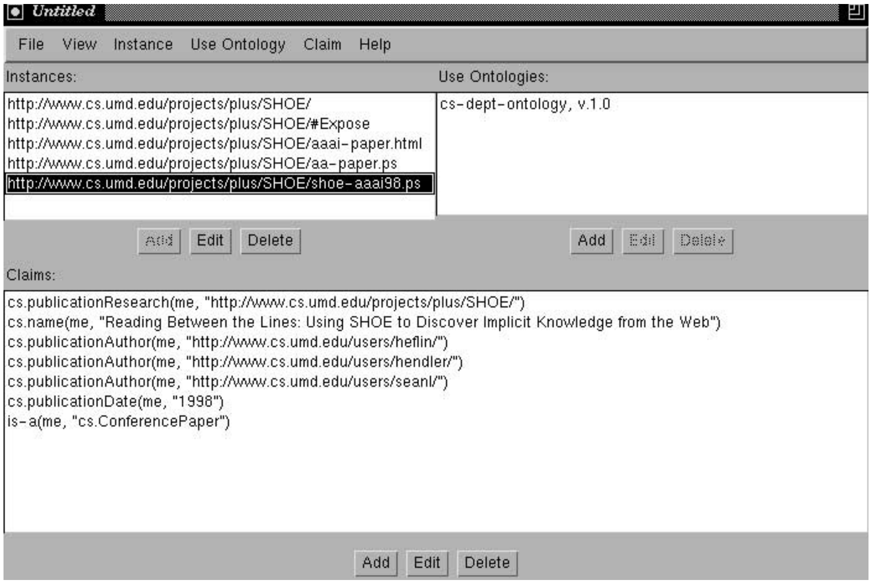
An author can formally specify a Semantic Web vocabulary using an ontology or a schema. Such ontologies and schemas are also typically sharable (so users can agree to use the same definitions) and extensible (so users can agree on some common set of definitions but add terms and definitions as necessary). Researchers expect that ontology hierarchies will develop, with top-level abstract ontologies at the root and domain-specific ontologies at the leaves. Thus, automatic interoperability between a pair of ontologies exists to the degree that they share a common ancestor. The language's expressivity determines the potential richness of an ontology's definitions. Most languages let ontologies define class taxonomies so that it is possible to say, for example, a car is a type of vehicle. They also allow for the definition of properties for each class and relationships between multiple classes. Some languages might also allow the formation of more complex definitions, using axioms from some form of logic.

Major differences exist between the leading Semantic Web languages. The resource description framework (RDF) Schema[1] is the least expressive. 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. Simple HTML Ontology Extensions (SHOE)[2] is based on a frame system but also allows Horn clause axioms (essentially, simple if–then rules), which authors can use 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] The Ontology Inference Layer (OIL), based on a frame system augmented with description logic, lets authors express different kinds of definitions.[4] The Defense Advanced Research Projects Agency Agent Markup Language (DAML) is a language under development with the intent to combine the best features of RDF, SHOE, and OIL.

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

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, Calif., 2000, pp. 443–449.
4. S. Decker et al., "Knowledge Representation on the Web," *Proc. 2000 Int'l Workshop on Description Logics (DL2000)*, Sun SITE Central Europe (CEUR), Aachen, Germany, 2000, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-33/.



```
http://www.cs.umd.edu/projects/plus/SHOE/
http://www.cs.umd.edu/projects/plus/SHOE/#Expose
http://www.cs.umd.edu/projects/plus/SHOE/aaai-paper.html
http://www.cs.umd.edu/projects/plus/SHOE/aa-paper.ps
http://www.cs.umd.edu/projects/plus/SHOE/shoe-aaai98.ps
```

```
cs-dept-ontology, v.1.0
```

```
cs.publicationResearch(me, "http://www.cs.umd.edu/projects/plus/SHOE/")
cs.name(me, "Reading Between the Lines: Using SHOE to Discover Implicit Knowledge from the Web")
cs.publicationAuthor(me, "http://www.cs.umd.edu/users/heflin/")
cs.publicationAuthor(me, "http://www.cs.umd.edu/users/hendler/")
cs.publicationAuthor(me, "http://www.cs.umd.edu/users/seanl/")
cs.publicationDate(me, "1998")
is-a(me, "cs.ConferencePaper")
```

Figure 1. The Knowledge Annotator. Here, the interface is being used to view semantic markup about a Simple HTML Ontology Extensions (SHOE) publication.

group's Web site. Most used the Knowledge Annotator, but some preferred a text editor. Although we 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

For semantic markup to be really useful, it needs to be ubiquitous, but using an authoring tool to generate a lot of markup is tedious. Detractors of the Semantic Web language approach often cite the difficulty of producing markup as the main reason it won't work. Fortunately, there are automatic and semiautomatic approaches for generating semantic markup.

Running SHOE. Some Web pages have regular structure, with labeled fields, lists, and tables. Often, an analyst can map these structures to an ontology and write a program to translate portions of the Web page into the semantic markup language. We developed Running SHOE (see 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 the program can skip header information) and end of a list (so that it can ignore trailing information); the start and 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 from different people describes the same entity. If we are to integrate descriptions about such an entity, we must use a common identifier (or key) when referring to it. A URL can often serve as this key because it identifies exactly one Web page, which a single person or organization owns. The regular pages that work best with Running SHOE tend to have lists of things, and each item in each list typically contains a hyperlink to a thing's homepage. 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 the program extracts the data, it expands all relative URLs 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 code can often cause the program to extract fields or records improperly; this table lets the user 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, because Running SHOE lets users save and retrieve templates, it is easy to regenerate new SHOE markup if the page's content changes.

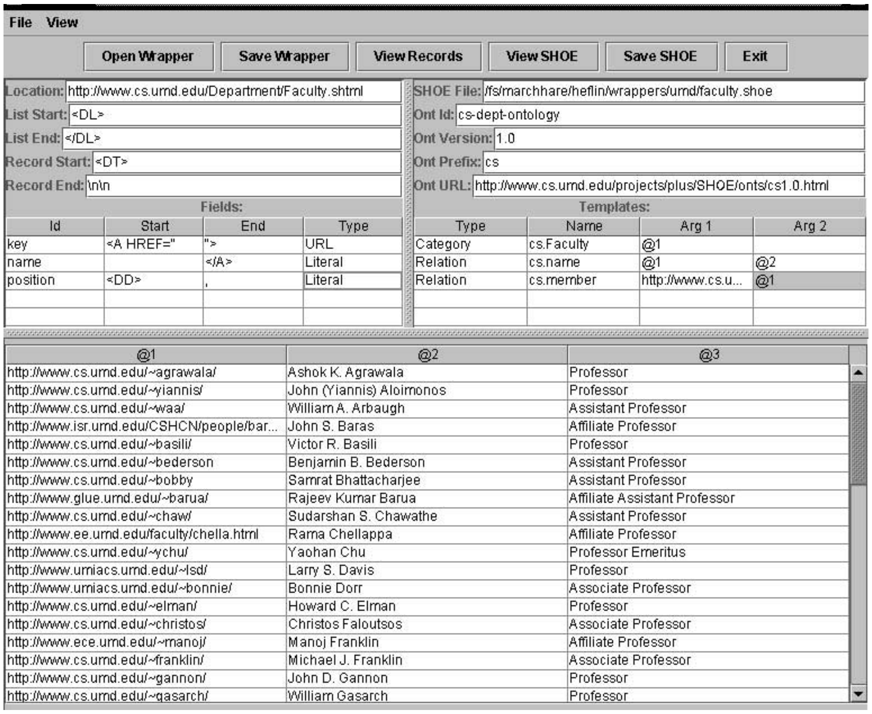
Computer science department Web sites often have faculty, project, course, and user lists that have ideal formats for Running SHOE. Each item in each list contains an <A> tag that provides the URL of the item's homepage, and this element's content is often the name of the entity being linked to, providing us with a value for the "name" relation. Other properties of the instance often follow and are delimited by punctuation, emphasis, or special spacing. With this tool, a single user can create SHOE markup about the faculty, courses, and projects of 15 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 computer science 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. This multistep process prevents Running SHOE from extracting markup from the CiteSeer site.

Publication SHOE Maker. To extract SHOE from CiteSeer, we built a tool called Publication SHOE Maker. PSM issues a query to get publications likely to be from a particular institution and retrieves a 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. Each publication page's layout is very similar, so PSM can extract the values of the desired fields easily.

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, which HomePageSearch (http://hpsearch.uni-trier.de) generates for each author. By using these URLs (as opposed to the authors' names), PSM can establish links to the appropriate instances.

Running SHOE and PSM are only two



Location: http://www.cs.umd.edu/Department/Faculty.shtml

List Start: <DL>

List End: </DL>

Record Start: <DT>

Record End: \n\n

SHOE File: /fs/marchhare/heflin/wrappers/umd/faculty.shoe

Ont Id: cs-dept-ontology

Ont Version: 1.0

Ont Prefix: cs

Ont URL: http://www.cs.umd.edu/projects/plus/SHOE/onts/cs1.0.html

Fields:

Id	Start	End	Type
key	<A HREF="	">	URL
name		</A>	Literal
position	<DD>	,	Literal

Templates:

Type	Name	Arg 1	Arg 2
Category	cs.Faculty	@1	
Relation	cs.name	@1	@2
Relation	cs.member	http://www.cs.u...	@1

@1	@2	@3
http://www.cs.umd.edu/~agrawala/	Ashok K. Agrawala	Professor
http://www.cs.umd.edu/~yiannis/	John (Yiannis) Aloimonos	Professor
http://www.cs.umd.edu/~waa/	William A. Arbaugh	Assistant Professor
http://www.isr.umd.edu/CSHCN/people/bar...	John S. Baras	Affiliate Professor
http://www.cs.umd.edu/~basili/	Victor R. Basili	Professor
http://www.cs.umd.edu/~bederson	Benjamin B. Bederson	Assistant Professor
http://www.cs.umd.edu/~bobby	Samrat Bhattacharjee	Assistant Professor
http://www.glue.umd.edu/~barua/	Rajeev Kumar Barua	Affiliate Assistant Professor
http://www.cs.umd.edu/~chaw/	Sudarshan S. Chawathe	Assistant Professor
http://www.ee.umd.edu/faculty/chella.html	Rama Chellappa	Affiliate Professor
http://www.cs.umd.edu/~ychu/	Yaohan Chu	Professor Emeritus
http://www.umiacs.umd.edu/~lsd/	Larry S. Davis	Professor
http://www.umiacs.umd.edu/~bonnie/	Bonnie Dorr	Associate Professor
http://www.cs.umd.edu/~elman/	Howard C. Elman	Professor
http://www.cs.umd.edu/~christos/	Christos Faloutsos	Associate Professor
http://www.ece.umd.edu/~manoj/	Manoj Franklin	Affiliate Professor
http://www.cs.umd.edu/~franklin/	Michael J. Franklin	Associate Professor
http://www.cs.umd.edu/~gannon/	John D. Gannon	Professor
http://www.cs.umd.edu/~gasarch/	William Gasarch	Professor

Figure 2. Running SHOE. A user can specify delimiters for recognizing fields and records, verify that they are extracted correctly, then create templates that translate the data into SHOE format.

examples of tools with which authors can generate markup. Other extraction tools might include machine-learning[4,5] or natural-language-processing techniques. As Extensible Markup Language becomes ubiquitous on the Web, generating wrappers will become easier, and authors will be able to use style sheets to transform a simple XML vocabulary into a semantically enriched one.

If a Web page's provider is willing to include semantic markup, the process can be even easier. For example, databases hold much of the Web's data, and scripts produce Web pages from that data. Because databases are structured resources, an analyst can determine the semantics of a database schema, map the schema to an ontology, and modify the scripts that produce the Web pages to include the appropriate semantic markup.

Integrating resources

After authors have described a number of diverse Web sites with semantic markup, the next problem is determining how to integrate the information. Information integration systems, such as Ariadne,[6] can be useful when developing an application that combines data from a finite number of predetermined sources, but are less helpful when integrating information "on the fly" is necessary. One solution mirrors the operation of contemporary search engines by crawling the Web and storing the information in a central repository.

Exposé

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 schedule when it will load that page. Thus, the cost function determines a traversal order. We assume SHOE pages will tend to be localized and interconnected. Therefore, we use a cost function that increases with distance from the start node, where paths through nonSHOE 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 parses each Web page, and if a page references an ontology that Exposé is unfamiliar with, it loads the ontology also. To update its list of pages to visit, Exposé identifies all of the hypertext links, category instances, and relation arguments within the page and evaluates each new URL as we discussed. Finally, the agent stores SHOE category and relation statements and any new ontology information in a knowledge base.

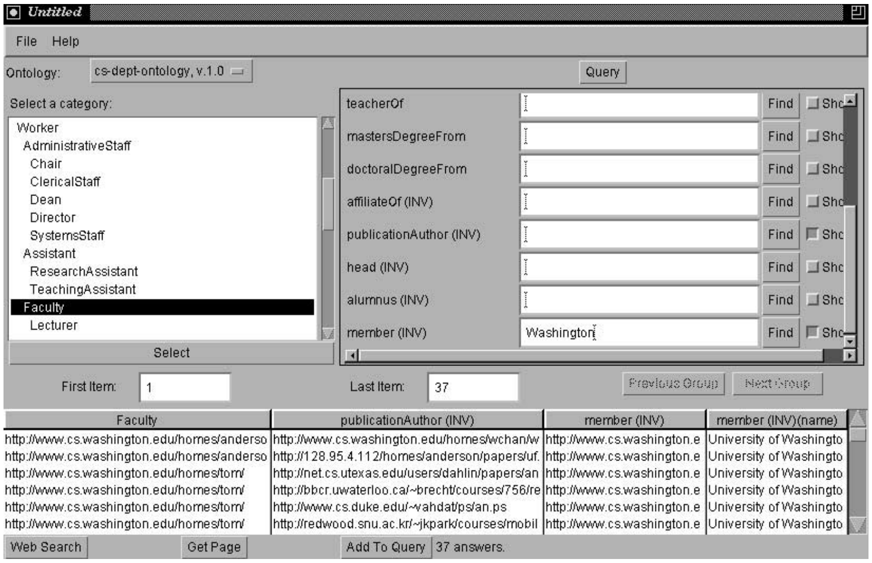
This KB will determine the system's query capabilities, and thus we must choose an appropriate knowledge representation system. Our SHOE tools all use a generic application programming interface for interaction with the KB, letting us easily use different backends. We have implemented versions of this API for Parka, a high-performance frame system;[7] XSB, a deductive database;[8] and Open Knowledge Base Connectivity-compliant KBs.[9]

By changing the back-end knowledge representation system, we get varying trade-offs between query response time and the degree to which the system uses inference to compute answers. For example, Parka answers 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, so it cannot use the extra Horn clause rules that SHOE allows. However, XSB can reason with these rules but is not as efficient as Parka. Alternately, the KB could be a relational or object database, providing the greatest scalability and best query response times but sacrificing the ability to infer additional answers. Clearly, the choice of the back-end system depends on the application's expected query needs.

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. Technically we did not need a Web crawler for our example, because we knew the locations of all the relevant pages a priori. However, in an ideal Semantic Web situation, the markup is the product of many individuals working independently, and users could not easily locate it 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 letting them submit structured queries and open documents by clicking on the URLs in the results. The user first chooses an ontology against which to



Ontology: cs-dept-ontology, v.1.0

Select a category:

Worker
AdministrativeStaff
Chair
ClericalStaff
Dean
Director
SystemsStaff
Assistant
ResearchAssistant
TeachingAssistant
Faculty
Lecturer

teacherOf		Find	☐ Sho
mastersDegreeFrom		Find	☐ Sho
doctoralDegreeFrom		Find	☐ Sho
affiliateOf (INV)		Find	☐ Sho
publicationAuthor (INV)		Find	☑ Sho
head (INV)		Find	☐ Sho
alumnus (INV)		Find	☐ Sho
member (INV)	Washington	Find	☑ Sho

First Item: 1

Last Item: 37

Faculty	publicationAuthor (INV)	member (INV)	member (INV)(name)
http://www.cs.washington.edu/homes/anderso	http://www.cs.washington.edu/homes/wchan/w	http://www.cs.washington.e	University of Washingto
http://www.cs.washington.edu/homes/anderso	http://128.95.4.112/homes/anderson/papers/uf.	http://www.cs.washington.e	University of Washingto
http://www.cs.washington.edu/homes/tom/	http://net.cs.utexas.edu/users/dahlin/papers/an	http://www.cs.washington.e	University of Washingto
http://www.cs.washington.edu/homes/tom/	http://bbcr.uwaterloo.ca/~brecht/courses/756/re	http://www.cs.washington.e	University of Washingto
http://www.cs.washington.edu/homes/tom/	http://www.cs.duke.edu/~vahdat/ps/an.ps	http://www.cs.washington.e	University of Washingto
http://www.cs.washington.edu/homes/tom/	http://redwood.snu.ac.kr/~jkpark/courses/mobil	http://www.cs.washington.e	University of Washingto

37 answers.

Figure 3. SHOE Search. With this tool, a user issues a query by choosing an ontology, choosing a category from that ontology, and then filling in desired values for properties that instances of that category might have.

issue the query (which essentially establishes a context for the query).

The user then chooses the desired object's class from a hierarchical list and is presented with a list of all properties that could apply to that object. After entering desired values for one or more of these properties, the user issues a query and receives a set of results in a table. If the user double-clicks on a binding that is a URL, the corresponding Web page will open in a new browser window. Thus, the user can browse the Semantic Web.

If SHOE markup does not describe all of the relevant Web pages, SHOE Search's standard query method will not be able to return an answer or might 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 of a number of popular search engines. Using SHOE Search in this way has two advantages over using the search engines directly:

- By prompting the user for values of properties, it increases the chance that the user will provide distinguishing information for the desired results.
- By automatically creating the query, it can exploit helpful features that users often overlook, such as quoting phrases or using the plus sign to indicate a mandatory term.

We build a query string that comprises a quoted short name for the selected category and, for each property value that the user specifies, a short phrase describing the property. The user's value, which we quote and precede with a plus sign to indicate that it is a mandatory phrase, follows the phrase describing the property.

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

Sample queries to the KB exposed one problem with the system: Sometimes it didn't integrate information from a department Web page and CiteSeer as expected. 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 because of multiple host names for a given IP address, default pages for a directory URL, host-dependent shortcuts such as a tilde for the users directory, symbolic links within the host, and so on. Additionally, some individuals might have multiple URLs that make equally valid keys for them, such as the URLs of both professional and personal homepages. 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 created a search engine called Semantic Search that is based on the technologies we describe. Semantic Search uses the SHOE Search tool as a query interface and provides utilities for authors, including links to an ontology library, the Knowledge Annotator, an online SHOE validation form, and a form for submitting new pages to the repository. We encourage readers to add markup to their own Web pages and submit them. Semantic Search is available at http://www.cs.umd.edu/projects/plus/SHOE/search/.

We have described a simple architecture for a Semantic Web system that parallels the way contemporary Web tools and search engines work. As Figure 4 shows, authors use various tools to add markup to Web pages, and a Web crawler discovers the information and stores it in a repository, which other tools can then query. Generally, authors need not produce all markup by hand; in many cases, simple extraction tools can generate accurate markup with minimal human effort. Although the tools that comprise this architecture are designed for use with the SHOE language, developers can create similar tools for other Semantic Web languages. Because any number of tools can produce and process the semantic markup on a Web page, other architectures are also possible. For example, developers could create an agent that queries pages directly as opposed to issuing queries to a Web-crawler-constructed repository.

If we achieve the Semantic Web vision, locating useful information on the Internet will be easier, and integrating diverse resources will be simpler. The first step is to design languages that we can use to express explicit semantics. The next step is to improve the systems and tools we describe, so users can naturally provide and receive information on the Semantic Web. Obviously, we must still overcome some obstacles: We need better schemes for ensuring interoperability between independently developed ontologies and approaches for determining who and what to trust. However,

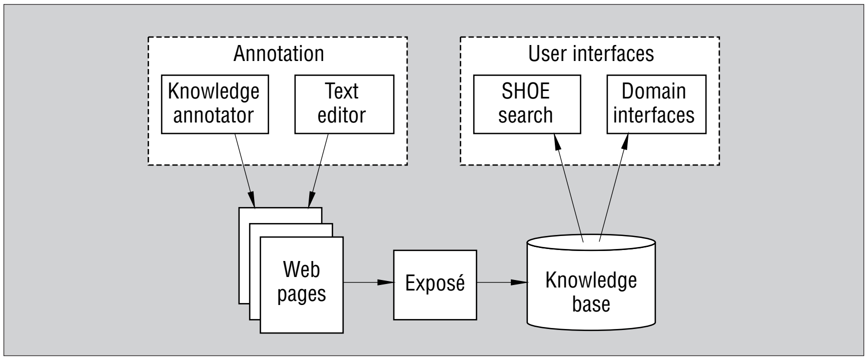


Figure 4. A simple Semantic Web system based on the tools we describe.

these challenges do not appear to be insurmountable, and the Semantic Web could be just around the corner. ■

Acknowledgments

The US Air Force Research Laboratory supported this work 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, pp. 59–66.

3. 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, Calif., 1998, pp. 36–42.

4. 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, Calif., 1998, pp. 517–523.

5. 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.

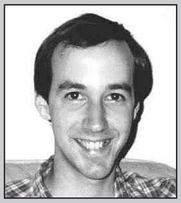
6. C. Knoblock et al., "Modeling Web Sources for Information Integration," *Proc. 15th Nat'l Conf. AI (AAAI-98)*, MIT–AAAI Press, Menlo Park, Calif., 1998, pp. 211–218.

7. 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, Calif., 1997.

8. K. Sagonas, T. Swift, and D. Warren, "XSB as an Efficient Deductive Database Engine," *Proc. 1994 ACM SIGMOD Int'l Conf. Management of Data (SIGMOD'94)*, ACM Press, New York, 1994, pp. 442–453.

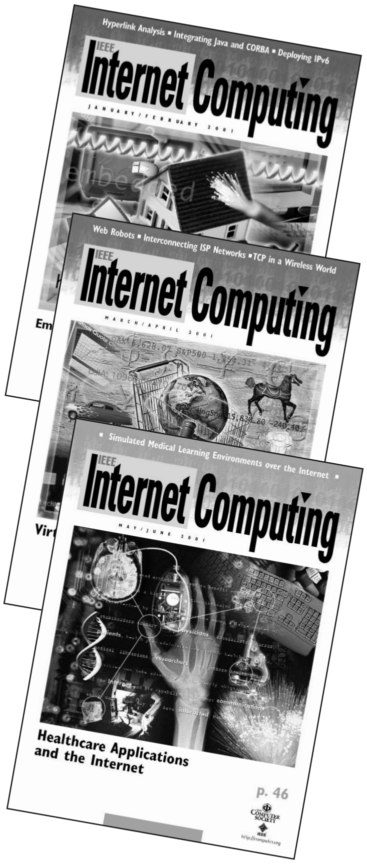
9. V. Chaudhri et al., "OKBC: A Programmatic Foundation for Knowledge Base Interoperability," *Proc. 15th Nat'l Conf. AI (AAAI-98)*, MIT–AAAI Press, Menlo Park, Calif., 1998, pp. 600–607.

The Authors



Jeff Heflin is a PhD candidate in the Computer Science Department at the University of Maryland. His research interests include Semantic Web languages, ontologies, Internet agents, and knowledge representation. He has worked in the computer consulting industry for four years as a data modeler, database designer, and database administrator. He received a BS in computer science from the College of William and Mary and an MS in computer science from the University of Maryland. He is currently a member of the Joint US–EU Ad hoc Agent Markup Language Committee. Contact him at the University of Maryland, Dept. of Computer Science, College Park, MD 20742; heflin@cs.umd.edu.

James Hendler's biography appears on p. 37.


Hyperlink Analysis • Integrating Java and CORBA • Deploying IPv6
IEEE Internet Computing
JANUARY/FEBRUARY 2001
Web Robots • Interconnecting ISP Networks • TCP in a Wireless World
IEEE Internet Computing
MARCH/APRIL 2001
• Simulated Medical Learning Environments over the Internet •
IEEE Internet Computing
MAY/JUNE 2001
Healthcare Applications and the Internet
p. 46