Semantic Interoperability on the Web

XML will have a profound impact on the way data is exchanged on the Internet. An important feature of this language is the separation of content from presentation, which makes it easier to select and/or reformat the data. However, due to the likelihood of numerous industry and domain specific DTDs, those who wish to integrate information will still be faced with the problem of semantic interoperability. In this paper we discuss why this problem is not solved by XML, and then discuss why the Resource Description Framework is only a partial solution. We then present the SHOE language, which we feel has many of the features necessary to enable a semantic web, and describe an existing set of tools that make it easy to use the language.

Jeff Heflin, University of Maryland
James Hendler, University of Maryland

1. Introduction

XML (the Extensible Markup Language) will have a profound impact on the way data is exchanged on the Internet. An important feature of this language is the separation of content from presentation, which makes it easier to select and/or reformat the data. However, if the growing proliferation of DTDs is indicative, then web developers will still be faced with the problem of semantic interoperability, i.e., the difficulty in integrating resources that were developed using different vocabularies and different perspectives on the data. To achieve semantic interoperability, systems must be able to exchange data in such a way that the precise meaning of the data is readily accessible and the data itself can be translated by any system into a form that it understands.

The benefits of semantic interoperability would be numerous. For example, search can often be frustrating because of the limitations of keyword-based matching techniques. Users frequently experience one of two problems: they either get back no results or too many irrelevant results. The problem is that words can be synonymous (that is, two words have the same meaning) or polysemous (a single word has multiple meanings). However, if the languages used to describe web pages were semantically interoperable, then the user could specify a query in the terminology that was most convenient, and be assured that the correct results were returned, regardless of how the data was expressed in the sources.

Intelligent internet agents could also benefit from semantic interoperability. Currently, such agents are very sensitive to changes in the format of a web page. Although these agents would not be affected by presentation changes if the pages were available in XML, they would still break if the XML representation of the data was changed slightly (for example if the element **`<PRICE>`** was renamed to **`<UNITPRICE>`**). Semantic interoperability would allow agents to be more robust. A useful function for internet agents would be the ability to automatically integrate information from multiple sources. For example, a travel-planning agent might need to access data from airlines, hotels, rental car companies, weather sites, and tourist sites, each of which may have different ways of representing the relevant data. Such an agent would be faced with the problem of translating the different vocabularies and representations for this data into a common format that it could work with.

We begin this paper by examining why XML alone is insufficient for achieving semantic interoperability. We then discuss RDF (the Resource Description Framework), a recommendation by the W3C (World Wide Web Consortium) which can be used to solve some of the problems of semantic interoperability. However, we argue that RDF falls short of this goal, and present SHOE (Simple HTML Ontology Extensions), a language that was expressly designed for this purpose.

2. XML and DTDs

A DTD (document type definition) can be used to ensure that XML documents conform to a common grammar. Thus a DTD provides a syntax for an XML document, but the semantics of a DTD are implicit. That is, the meaning of an element in a DTD is either inferred by a human due to the name assigned to it, is described in a natural-language comment within the DTD, or is described in a document separate from the DTD. Humans can then build these semantics into tools that are used to interpret or translate the XML documents, but software tools cannot acquire these semantics independently. Thus, an exchange of XML documents works well if the parties involved have agreed to a DTD beforehand, but becomes problematic when one wants to search across the entire set of DTDs or to spontaneously integrate information from multiple sources.

One of the hardest problems in any integration effort is mapping between different representations of the same concepts — the problem of integrating DTDs is no different. One difficulty is identifying and mapping differences in naming conventions. As with natural language, XML DTDs have the problems of polysemy and synonymy. For example, the elements **<PERSON>** and **<INDIVIDUAL>** might be synonymous. Similarly, an element such as **<SPIDER>** might be polysemous: in one document it could mean a piece of software that crawls the World Wide Web while in another it means an arachnid that crawls a web of the silky kind. In general, it is difficult for machines to make determinations of this nature, even if they have access to a complete automated dictionary and thesaurus. Naming problems such as these are not restricted to element names, the same issues apply to attribute names.

An even more difficult problem is identifying and mapping differences in structure. XML's flexibility gives DTD authors a number of choices. Designers attempting to describe the same concepts may choose to do it in many different ways. In Figure 1, three possible representations of a person's name are shown. One choice involves whether the name is a string or is an element with structure of its own. Another choice is whether the name is an attribute or an element. One of the reasons for these problems is the lack of semantics in XML. There is no special meaning associated with attributes or content elements. Element content might be used to describe properties of an object or group related items, while attributes might be used to specify supplemental information or single-valued properties.

```
<!-- The NAME is a subelement with character content -->
<PERSON>
  <NAME>John Smith</NAME>
</PERSON>

<!-- The NAME is a subelement with element content -->
<PERSON>
```

```
  <NAME><FNAME>John</FNAME><LNAME>Smith</LNAME></NAME>
</PERSON>

<!-- The NAME is an attribute of PERSON -->
<PERSON NAME="John Smith">
```

Figure 1. Structural Differences in Representation

Once humans have identified the appropriate mappings between two DTDs, it is possible to write XSLT (XSL Transformations) stylesheets that can be used to automatically translate one document into the format of another. Although this is a good solution to the integration problem when only a few DTDs are relevant, it is unsatisfactory when there are many DTDs; if there are n DTDs, then there would need to be $O(n^2)$ different stylesheets to allow automatic transformation between any pair of them. Furthermore, when a DTD was created or revised, someone would have to create or revise the n stylesheets to transform it to all other DTDs. Obviously, this is not a feasible solution.

Of course, the problems of mapping DTDs would go away if we could agree on a single universal DTD, but even at the scale of a single corporation, data standardization can be difficult and time consuming — data standardization on a worldwide scale would be impossible. Even if a comprehensive, universal DTD was possible, it would be so unimaginably large that it would be unusable, and the size of the standards committee that managed it would preclude the possibility of extension and revision at the pace required for modern data processing needs.

3. RDF and RDF Schema

RDF [Lassila and Swick 99] is a recommendation endorsed by the W3C. One of the goals of RDF is "to specify semantics for data based on XML in a standardized interoperable manner." Technically, RDF is not a language, but a data model of metadata instances. The basic data model is very simple; it consists of nodes connected by labeled arcs, where the nodes represent web resources and the arcs represent properties of these resources. To include RDF in files, its designers have chosen an XML syntax although they emphasize this is only one of many possible representations of the RDF model. RDF has a number of abbreviated syntactical variations which is an advantage for content providers but requires more machinery in RDF parsers. Since these syntaxes all have a well-defined mapping into the data model, they avoid some of the problems with representational choices in basic XML. Nevertheless, it is still easy to create different representations for a concept.

To allow for the creation of controlled, sharable, and extensible vocabularies, the RDF working group has developed the RDF Schema Specification [Brickley and Guha 99]. This specification defines a number of properties that have specific semantics. The property **`rdf:type`** is used to express that a resource is a member of a given class, while the property **`rdfs:subClassOf`** essentially states that one class is a subset of another. These properties are equivalent to the INSTANCE-OF and IS-A links that have been used in AI (Artificial Intelligence) systems for decades. With the **`rdfs:subClassOf`** property, schema designers can build taxonomies of classes for organizing their resources. RDF Schema also provides properties for describing properties: the property **`rdfs:subPropertyOf`** allows properties to be specialized in a way similar to classes,

while the properties **rdfs:domain** and **rdfs:range** allow constraints to be placed on the domain and range of a property.

Since it is possible that different schemas may use the same strings to represent different conceptual meanings, RDF uses XML namespaces to assign a separate namespace to each schema. This approach has two disadvantages. First since namespaces can be used with any element and RDF schemas need not be formally specified, it is possible to write RDF statements such that it is ambiguous as to whether certain tags are RDF or intermeshed tags from another namespace. Second, each RDF section must explicitly specify the namespace for every schema that is referenced in that section, even for schemas that were extended by a schema whose namespace has already been specified.

In RDF, schemas are extended by simply referring to objects from that schema as resources in a new schema. Since schemas are assigned unique URIs (Uniform Resource Identifiers), the use of XML namespaces guarantees that exactly one object is being referenced. Unfortunately, RDF does not have a feature that allows local aliases to be provided for properties and classes; such a feature is necessary because achieving consensus for schema names will be impossible on a worldwide scale. Although an alias can be approximated using the **rdfs:subClassOf** or **rdfs:subPropertyOf** properties to state that the new name is a specialization of the old one, there is no way to state an equivalence. This can be problematic if two separate schemas "rename" a class; since both schemas have simply subclassed the original class, the information that all three classes are equivalent is lost.

RDF schema is written entirely in RDF statements. Although at first this may seem like elegant bootstrapping, closer inspection reveals that it is only a reuse of syntax. RDF is not expressive enough to define the special properties **rdfs:subClassOf**, **rdfs:subPropertyOf**, **rdfs:domain** and **rdfs:range**, and thus correct usage of these properties must be built into any tool that processes RDF with RDF schemas. However, a very subtle but potentially dangerous problem is hidden here. The definition of a class (or property) is a collection of RDF statements about the same resource that use properties from the RDFS namespace. Typically, these statements appear on a single web page, grouped using a **rdf:Description** element. However, since a resource is identified by a URI, there is no reason why some of these statements could not appear on another page. Thus anyone could add to the definition of an object introduced in another schema. Although there are many situations where this is beneficial, accidental or malicious definitions may alter the semantics in an undesirable way. For example, someone could make the class **WebDeveloper** a subclass of **OverpaidPerson**, and anyone who stated that they were a **WebDeveloper**, would now also be implicitly stating they were an **OverpaidPerson**. To resolve such problems one would have to add to RDF the ability for a document to specify which other documents (including schemas) it is consistent with.

RDF does not possess any mechanisms for defining general axioms (rules that allow additional reasoning). For example, one could not specify that the **subOrganization** property is transitive or that the **parentOf** and **childOf** properties are inverses of each other. In logic, axioms are used to constrain the possible meaning of a term and thus provide stronger semantics. However, there are other benefits to the use of axioms: an axiom can provide additional information that was not explicitly stated and, perhaps more importantly for distributed systems such as the Web, axioms can be used to map between different representations of the same concepts.

Another potential problem for RDF is the Web's tendency towards rapid change. Although RDF provides a method for revising schemas, this method is insufficient. Essentially each new version of a schema is given its own URI and thus can be thought of as a distinct schema in and of itself. However, the version is really just a schema that extends the original version; its only link to the original schema is by use of the **`rdfs:subClassOf`** and **`rdfs:subPropertyOf`** properties to point to the original definitions of each class and property. As such, a true equivalence is not established between the items. Additionally, if schemas and resources that refer to the schema that was updated wish to reflect the changes, they must change every individual reference to a schema object to use the new URI. Finally, since schemas do not have an official version associated with them, there is no way to track the revisions of a schema unless the schema maintainer uses a consistent naming scheme for the URIs.

4. SHOE

SHOE is an ontology-based knowledge representation language designed for the Web [Luke et al. 97] [Heflin, Hendler and Luke 99]. The current version of the language is the result of an effort that started in 1996 and which anticipated many of the features that were subsequently added to XML and RDF. SHOE uses knowledge oriented elements, and associates meaning with content by making each web page commit to one or more ontologies. The term ontology is used by AI researchers to mean a formal specification of the concepts and relations of some domain. Thus an RDF schema can be considered an ontology, although it is more limited than most AI ontologies. SHOE ontologies permit the discovery of implicit knowledge through the use of taxonomies and inference rules, allowing content providers to encode only the necessary information on their web pages, and to use the level of detail that is appropriate to the context. Interoperability is promoted through the sharing and reuse of ontologies.

4.1 The Language

The SHOE syntax is defined as an application of SGML (Standard Generalized Markup Language) that extends the HTML DTD. This choice was primarily because XML was still evolving when the syntax was originally designed. However, due to the similarities between SGML and XML, it was easy to create a slight variant of the syntax that is compatible with XML. Since most web sites are still written in HTML (which is not compatible with XML), the SGML version of SHOE remains the standard for implementation, but web sites that have begun to migrate to XML (by using XHTML or another XML application) can use the SHOE XML DTD. This DTD is reproduced in the Appendix.

SHOE ontologies are made publicly available by locating them on web pages. An **`<ontology>`** tag delimits the machine-readable portion of the ontology and specifies a unique identifier and a version number for the ontology. An ontology can define categories, relations, and other components.

Categories, which are similar to RDF classes, are introduced with a **`<def-category>`** tag and may specify one or more subsuming categories. Categories allow taxonomies to be built from the top down by subdividing known classes into smaller sets. SHOE does not allow subsuming categories to be specified for a category defined elsewhere, effectively preventing anyone from arbitrarily making **`WebDeveloper`** a subclass of **`OverpaidPerson`** as can be done with RDF.

Relations are similar to RDF properties in that they are used to describe objects of interest. However, while RDF properties are binary, SHOE relations are n-ary predicates. They are defined with a **<def-relation>** tag and must specify types for each argument.

As is common in many ontology efforts, such as Ontolingua [Farquhar, Fikes, and Rice 97] or Cyc [Lenat and Guha 90], SHOE ontologies build on or extend other ontologies, forming a lattice with the most general ontologies at the top and the more specific ones at the bottom. Ontology extension is expressed in SHOE with the **<use-ontology>** tag, which indicates the id and version number of an ontology that is extended. An optional **url** attribute allows systems to locate the ontology if needed and a **prefix** attribute is used to establish a short local identifier for the ontology. When an ontology refers to an element from an extended ontology, this prefix and a period is appended before the element's name. In this way, references are guaranteed to be unambiguous, even when two ontologies use the same term to mean different things. By chaining the prefixes, one can specify a path through the extended ontologies to an element whose definition is given in a more general ontology.

Sometimes an ontology may need to use a term from another ontology, but a different label may be more useful within its context. The **<def-rename>** tag allows the ontology to specify a local name for a concept from any extended ontology. This local name must be unique within the scope of the ontology in which the rename appears. Renaming allows domain specific ontologies to use the vocabulary that is appropriate for the domain, while maintaining interoperability with other domains.

SHOE uses inference rules, indicated by the **<def-inference>** tag, to supply additional axioms. A SHOE inference rule consists of a set of antecedents (one or more subclauses describing claims that entities might make) and a set of consequents (consisting of one or more subclauses describing a claim that may be inferred if all claims in the body are made). The **<inf-if>** and **<inf-then>** tags indicate the antecedents and consequents of the inference, respectively. There are three types of inference subclauses: category, relation and comparison. The arguments of any subclause may be a constant or a variable, where variables are indicated by the keyword **VAR**. Constants must be matched exactly and variables of the same name must bind to the same value.

Unlike RDF, SHOE makes a distinction between what can be said in an ontology and what can be said on an arbitrary web page. Ordinary web pages declare one or more instances that represent SHOE entities, and each instance describes itself using categories and relations. The syntax for instances includes an **<instance>** element that has an attribute for a **key** that uniquely identifies the instance. We recommend that the URL of the web page be used as this key, since it is guaranteed to identify only a single resource. An instance commits to a particular ontology by means of the **<use-ontology>** tag, which has the same function as the identically named element used within ontologies. To prevent ambiguity in the declarations, ontology components are referred to using the prefixing mechanism described earlier. The use of common ontologies makes it possible to issue a single logical query to a set of data sources and enables the integration of related domains. Additionally, by specifying ontologies the content author indicates exactly what meaning he associates with his claims, and does not need to worry that an arbitrary claim made by someone else will alter this meaning.

The primary means of achieving interoperability in SHOE is through use of the ontology extension and renaming features. Two categories are similar to the extent that they share the same supercategories. As a result, ontologies are interoperable to the extent that they share the

same ancestor ontologies. For example, in Figure 2, the term **`Spider`** means different things in **`internet-ont`** and **`bug-ont`** because the categories have different ancestors. However the term **`WebBot`** in **`internet-ont2`** means the same thing as **`Spider`** in **`internet-ont`** because a **`def-rename`** indicates that it is an alias of the term. Achieving interoperability in this manner depends on the ontologies being constructed correctly; therefore the most general ontologies (the ones that affect the most people) should be designed and maintained by experts. However, the nature of distributed environments will result in ontologies that are not full interoperable. In [Heflin and Hendler 2000] we discuss strategies for reintegrating such ontologies.

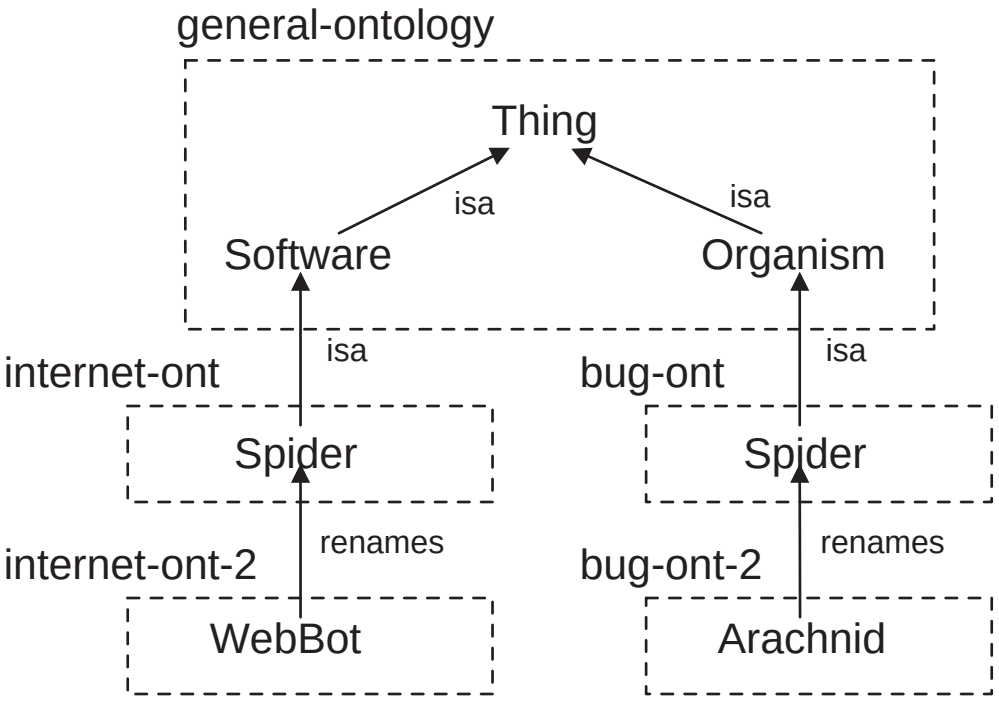

Figure 2. Ontology Interoperability

In this figure, *isa* refers to the presence of a **`def-category`** tag and *renames* refers to the presence of a **`def-rename`** tag.

Like RDF schemas, SHOE ontologies are revised by creating a new file. However, each version of a SHOE ontology is assigned a unique version number. When instances and other ontologies commit to a particular ontology, they also commit to a particular version. The **`ontology`** element includes an optional **`backward-compatible-with`** attribute to state which previous versions it is compatible with. Systems can then use this tag as a suggestion that it is safe to use the newer version of an ontology to reason about instances that committed to an earlier one. A more technical discussion of how we handle changing ontologies in SHOE can be found in [Heflin and Hendler 2000].

4.2 Implementation

In order to demonstrate the feasibility of SHOE, we built a number of tools to support its use. In this section, we describe this suite of tools and one scenario for their use. The architecture of this scenario is depicted in Figure 3. Demonstrations of most of these tools are available from the SHOE web site: http://www.cs.umd.edu/projects/plus/SHOE/

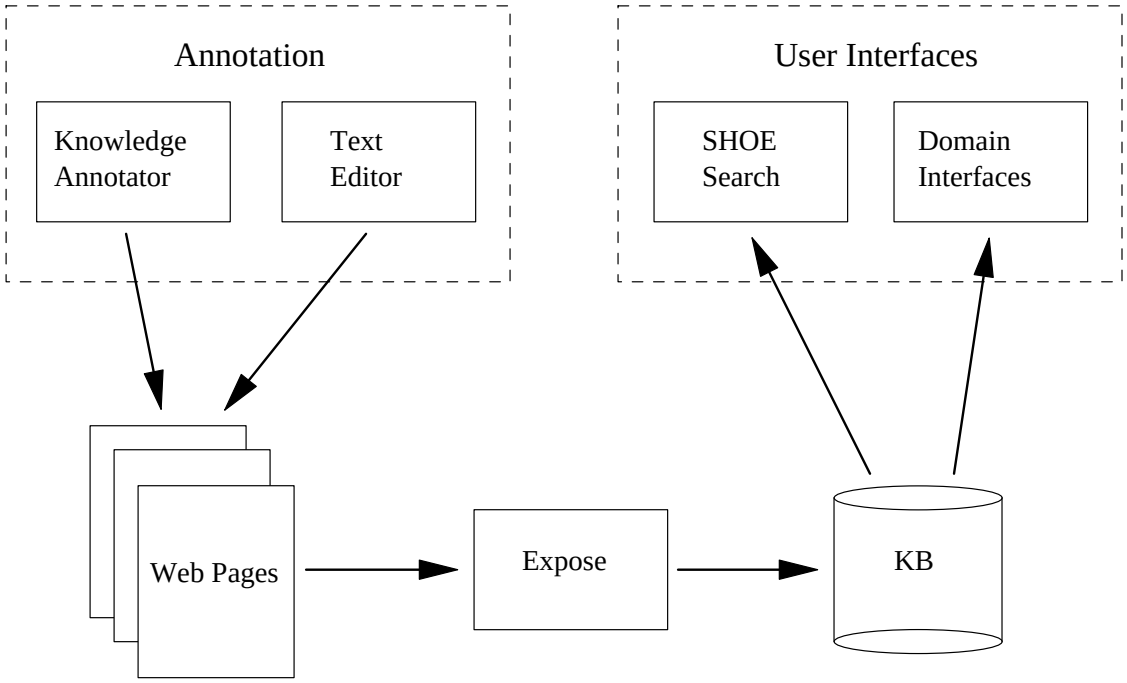

Figure 3. Architecture of a SHOE System

The first step in using SHOE is to add markup to the web pages, a process we call annotation. In order to annotate a web page, the user must select an appropriate ontology and then use that ontology's vocabulary to describe the concepts on the page. As with HTML, SHOE markup can be added to a page using a simple text editor. However, unlike HTML processors, SHOE processors are not very forgiving, and errors can result in large portions of the annotations being ignored. To ease the burden on the author, we have designed the Knowledge Annotator, a tool that makes it easy to add SHOE knowledge to web pages by making selections and filling in forms. As shown in Figure 4, this tool has an interface that displays instances, ontologies, and claims. 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 from a corresponding list. The available relations will automatically filter 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.

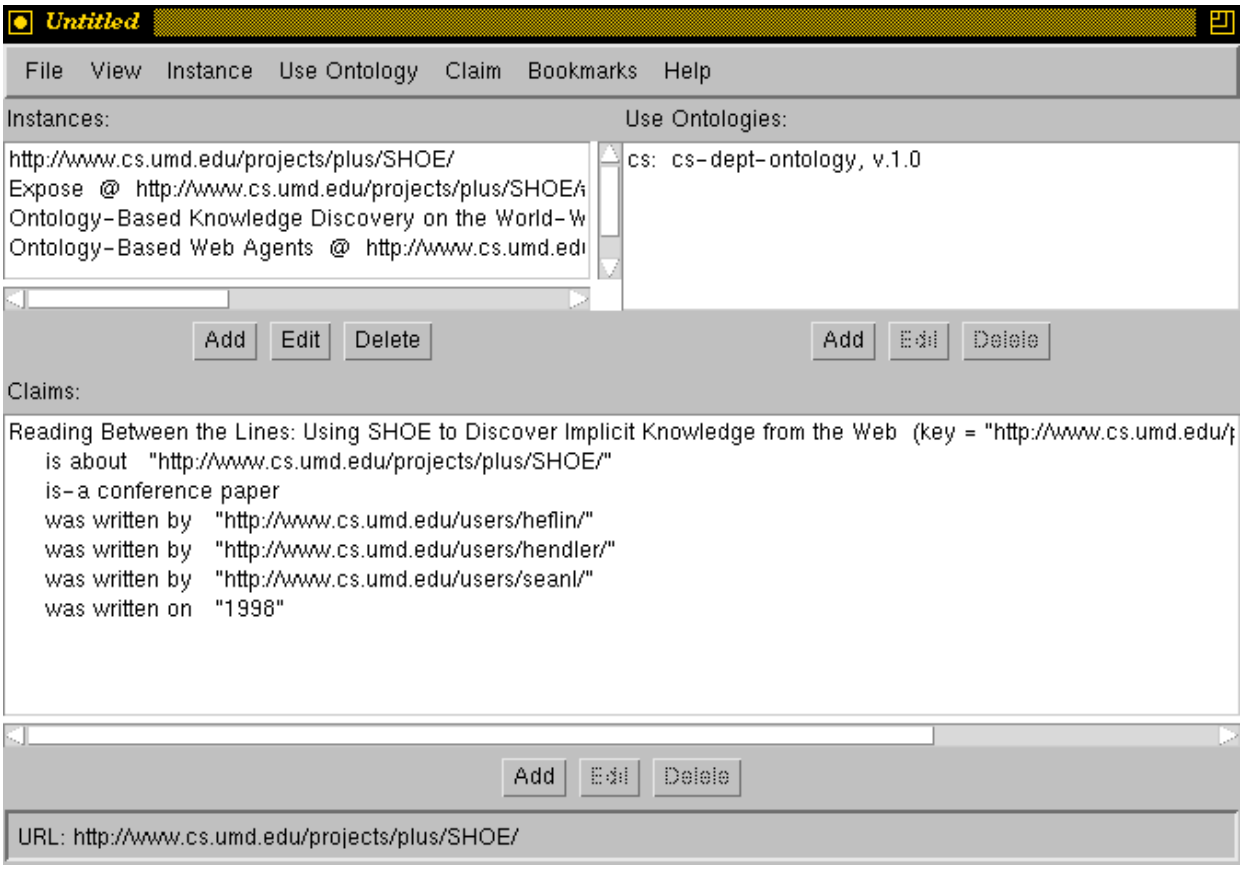

Figure 4. The Knowledge Annotator

When SHOE pages are annotated and placed on the Web, they can be queried and indexed. Our current system parallels the way modern search engines work, that is we collect the knowledge from the pages and store it in a repository. For this purpose, we have developed Exposé, a web-crawler that searches for web pages with SHOE mark-up 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 it 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. When Exposé loads a web page, it parses it, and if the web 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 claims, as well as any new ontology information, in a KB (knowledge base).

Currently, we store SHOE knowledge in a Parka KB [Stoffel, Taylor, and Hendler 97], but all of our tools are designed with a generic KB API, so that an alternate KR (knowledge

representation) system could be used with a minimum of effort. Considering the size of the Web, any KR system that is used must be scalable. This is why we use Parka, it provides a good tradeoff between the most common types of inferences for SHOE and query efficiency. Parka has been shown to answer queries on KBs with millions of assertions in seconds, and when used on parallel machines, it provides even better performance.

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, which is shown in Figure 5, 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 is presented with 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 be opened in a new window of the user's web browser.

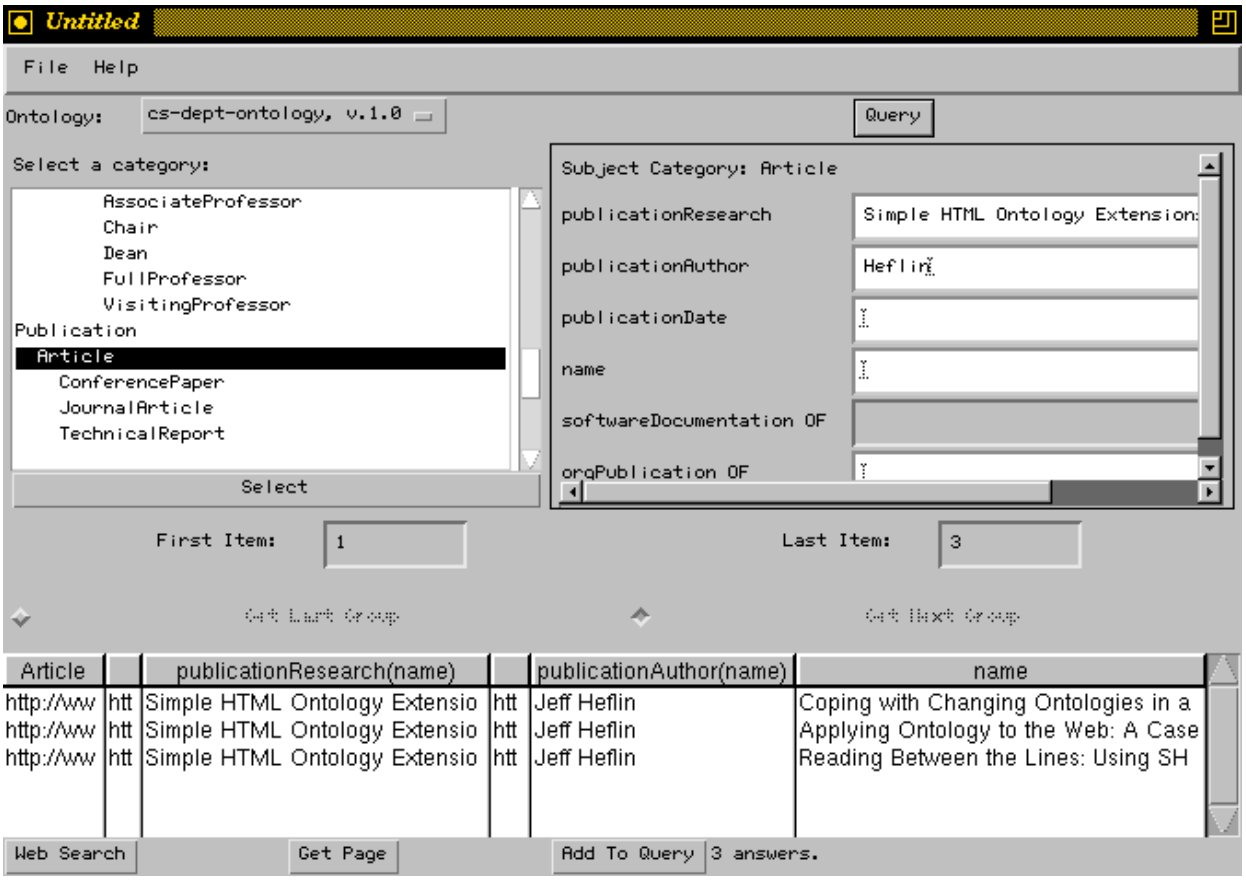

Figure 5. SHOE Search

Conclusion

SHOE is not the only AI language for the Web. The Ontobroker project [Fensel et al. 98] uses a language to describe data that is embedded in HTML, but relies on a centralized broker for ontology definitions. The Ontology Markup Language (OML) and Conceptual Knowledge Markup Language (CKML) [Kent 99] are used together for semantic markup that is based on the theories of formal concept analysis and information flow. However, both of the languages are basically web syntaxes for more traditional KR languages, and neither considers the special characteristics of the Web to the degree that SHOE does.

XML will revolutionize the Web, but semantic interoperability is needed to achieve the Web's true potential. We have discussed the limitations of XML and RDF with respect to semantic interoperability, and presented the SHOE language. In describing the basic elements of SHOE, we have explained how it is better suited for semantics on the Web than either XML DTDs or RDF. In order to demonstrate SHOE, we have built a suite of tools for its use and have described those tools here.

Appendix: SHOE XML DTD

```
<!ELEMENT shoe          (ontology | instance)* >

<!-- Declarations for ontologies -->
<!ELEMENT ontology      (use-ontology | def-category | def-relation |
                         def-rename | def-inference | def-constant |
                         def-type)* >

<!ATTLIST ontology
        id              CDATA   #REQUIRED
        version         CDATA   #REQUIRED
        description     CDATA   #IMPLIED
        declarators     CDATA   #IMPLIED
        backward-compatible-with        CDATA   #IMPLIED >

<!ELEMENT use-ontology   EMPTY >
<!ATTLIST use-ontology
        id              CDATA   #REQUIRED
        version         CDATA   #REQUIRED
        prefix          CDATA   #REQUIRED
        url             CDATA   #IMPLIED >

<!ELEMENT def-category   EMPTY >
<!ATTLIST def-category
        name            CDATA   #REQUIRED
        isa             CDATA   #IMPLIED
        description     CDATA   #IMPLIED
        short           CDATA   #IMPLIED >

<!ELEMENT def-relation   (def-arg)* >
<!ATTLIST def-relation
        name            CDATA   #REQUIRED
        short           CDATA   #IMPLIED
        description     CDATA   #IMPLIED >

<!ELEMENT def-arg        EMPTY >
<!ATTLIST def-arg
        pos             CDATA   #REQUIRED
        type            CDATA   #REQUIRED
        short           CDATA   #IMPLIED >
<!-- pos must be either an integer, or one of the strings: FROM or TO -->

<!ELEMENT def-rename     EMPTY >
<!ATTLIST def-rename
        from            CDATA   #REQUIRED
        to              CDATA   #REQUIRED >

<!ELEMENT def-constant   EMPTY >
<!ATTLIST def-constant
        name            CDATA   #REQUIRED
        category        CDATA   #IMPLIED >
```

```
<!ELEMENT def-type       EMPTY >
<!ATTLIST def-type
        name            CDATA   #REQUIRED
        description     CDATA   #IMPLIED
        short           CDATA   #IMPLIED >

<!-- Declarations for inferences -->
<!-- Inferences consist of if and then parts, each of which
     can contain multiple relation and category clauses -->
<!ELEMENT def-inference  (inf-if, inf-then) >
<!ATTLIST def-inference
        description     CDATA   #IMPLIED >
<!ELEMENT inf-if         (category | relation | comparison)+ >
<!ELEMENT inf-then       (category | relation)+ >
<!ELEMENT comparison     (arg, arg) >
<!ATTLIST comparison
        op      (equal | notEqual | greaterThan |
                 greaterThanOrEqual | lessThanOrEqual |
                 lessThan)      #REQUIRED >

<!-- Declarations for instances -->
<!ELEMENT instance       (use-ontology | category | relation | instance)* >
<!ATTLIST instance
        key             CDATA   #REQUIRED
        delegate-to     CDATA   #IMPLIED >

<!ELEMENT category       EMPTY >
<!ATTLIST category
        name            CDATA   #REQUIRED
        for             CDATA   #IMPLIED
        usage           (VAR | CONST)   "CONST" >
<!-- If VAR is specified for a category that is not within a <def-inference>,
        then it is ignored -->

<!ELEMENT relation       (arg)* >
<!ATTLIST relation
        name            CDATA   #REQUIRED >
<!ELEMENT arg            EMPTY >
<!ATTLIST arg
        pos             CDATA   #REQUIRED
        value           CDATA   #REQUIRED
        usage           (VAR | CONST)   "CONST" >

<!-- pos must be either an integer, or one of the strings: FROM or TO -->
<!-- If VAR is specified for an arg that is not within a <def-inference>,
        then it is ignored -->
```

Acknowledgements

This work was supported by the Army Research Laboratory under contract number DAAL01-97-K0135 and the Air Force Research Laboratory under grant F306029910013.

Bibliography

[Brickley and Guha 99] Brickley, D. and Guha, R. 1999. *Resource Description Framework (RDF) Schema Specification (Proposed Recommendation)*, W3C (World Wide Web Consortium) (1999). (At http://www.w3.org/TR/1999/PR-rdf-schema-19990303)

[Farquhar, Fikes, and Rice 97] Farquhar, A.; Fikes, R.; and Rice, J. Tools for Assembling Modular Ontologies in Ontolingua. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 436-441. Menlo Park, CA: AAAI/MIT Press (1997).

[Fensel et al. 98] Fensel, D.; Decker, S.; Erdmann, M.; and Studer, R. Ontobroker: How to enable intelligent access to the WWW. In *AI and Information Integration, Technical Report WS-98-14*, 36-42. Menlo Park, CA: AAAI Press (1998).

[Heflin, Hendler and Luke 99] Heflin, J.; Hendler, J.; and Luke, S. *SHOE: A Knowledge Representation Language for Internet Applications, Technical Report, CS-TR-4078 (UMIACS TR-99-71)*, Dept. of Computer Science, University of Maryland (1999).

[Heflin and Hendler 2000] Heflin, J. and Hendler, J. Dynamic Ontologies on the Web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000).*. Menlo Park, CA: MIT/AAAI Press (2000).

[Kent 99] Kent, R.E. Conceptual Knowledge Markup Language: The Central Core. In *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management* (1999).

[Lassila and Swick 99] Lassila, O. and Swick, R. *Resource Description Framework (RDF) Model and Syntax Specification*, W3C (World-Wide Web Consortium) (1999). (At http://www.w3.org/TR/1999/REC-rdf-syntax-19990222)

[Lenat and Guha 90] Lenat, D. and Guha, R. *Building Large Knowledge Based Systems.* Reading, Mass.: Addison-Wesley (1990).

[Luke et al. 97] Luke, S.; Spector, L.; Rager, D.; and Hendler, J. Ontology-based Web Agents. In *Proceedings of the First International Conference on Autonomous Agents*, 59-66. New York, NY: Association of Computing Machinery (1997).

[Stoffel, Taylor, and Hendler 97] Stoffel K., Taylor, M., and Hendler, J. Efficient Management of Very Large Ontologies. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 442-447. Menlo Park, CA: AAAI/MIT Press (1997).

Biography

Jeff Heflin
University of Maryland
Department of Computer Science
College Park MD 20742 USA
E-mail: heflin@cs.umd.edu
Web: http://www.cs.umd.edu/~heflin

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. His research interests include knowledge representation, ontologies, web languages and internet agents.

James Hendler
University of Maryland
Department of Computer Science
College Park MD 20742 USA
E-mail: hendler@cs.umd.edu
Web: http://www.cs.umd.edu/~hendler

Dr. Hendler is currently working at the Defense Advanced Research Projects Agency (DARPA) as a program manager. The opinions expressed in this paper are his 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.