Searching the Web with SHOE

Jeff Heflin and James Hendler

Department of Computer Science
University of Maryland
College Park, MD 20742
{heflin, hendler}@cs.umd.edu

Abstract

Although search engine technology has improved in recent years, there are still many types of searches that return unsatisfactory results. This situation can be greatly improved if web pages use a semantic markup language to describe their content. We have developed SHOE, a language for this purpose, and in this paper describe a scenario for how the language could be used by search engines of the future. A major challenge to this system is designing a query tool that can exploit the power of a knowledge base while still being simple enough for the casual user. We present the SHOE Search tool, which allows the user to specify a context for his or her query, and then uses the context to help the user build a query by example.

1. Introduction

The World Wide Web is an information resource with virtually unlimited potential. However, this potential is relatively untapped because locating relevant information can often be time consuming and fruitless. Search engines are constrained by the limitations of string matching and link analysis while directories such as Yahoo are constrained by the staff-hours that they can devote to cataloging the Web. However, if machines could "understand" the content of web pages, then searches with high precision and recall would be possible. Of course, accurate natural language processing is still not possible in general domains. However, the focus of the Extensible Markup Language (XML) on describing content rather than presentation opens up new possibilities. Still, XML is only a syntax and it is likely that many incompatible tag sets will evolve to describe similar concepts. Although XML promotes interoperability between organizations that agree on standards ahead of time, the divergent vocabularies will make it of little use for general searches on the Web. A semantic markup language is needed for these purposes.

We have defined the Simple HTML Ontology Extensions (SHOE) language, an application of SGML and XML that allows users to define extensible vocabularies and associate machine understandable meaning with them. These vocabularies are ontologies that consist of category and relation definitions, which can be augmented with additional axioms as desired. SHOE promotes interoperability by having all ontologies publicly available on the Web and allowing domain specific ontologies to be created by the process of ontology extension. SHOE also addresses the evolution of the Web by explicitly including versioning features in the language (Heflin and Hendler 2000). Rather than describe the SHOE language in detail here, we point the interested reader to the SHOE Specification (Luke and Heflin 1997).

In this paper, we specifically address how SHOE can be used to perform more effective web searches. We begin by describing how SHOE markup is added to pages, how this information is collected, and how it is stored. We then present SHOE Search, a general purpose query tool that allows users to query SHOE information with a minimal understanding of how it is structured.

2. Overview of the SHOE Process

In this section we discuss the key steps for use of the SHOE language. Since SHOE is essentially a common language that can be used to exchange Web data by any number of applications, many architectures are possible. Here we focus on the architecture that we are currently using for our projects; this architecture is shown in Figure 1.

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. In SHOE, each concept is called an instance and is assigned a key, typically the URL of the web page that best represents the concept. The user can then express that the concept is a member of a particular category or that it has certain relationships. These relationships may either involve literal values such as strings or numbers (as may be the case with the relations *name* or *height*) or can be with other instances, in which case the key of the related instance must be known. Since there are no guarantees about the accuracy of information on the Web, category and relation declarations such as these are referred to as claims.

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. This tool has an interface

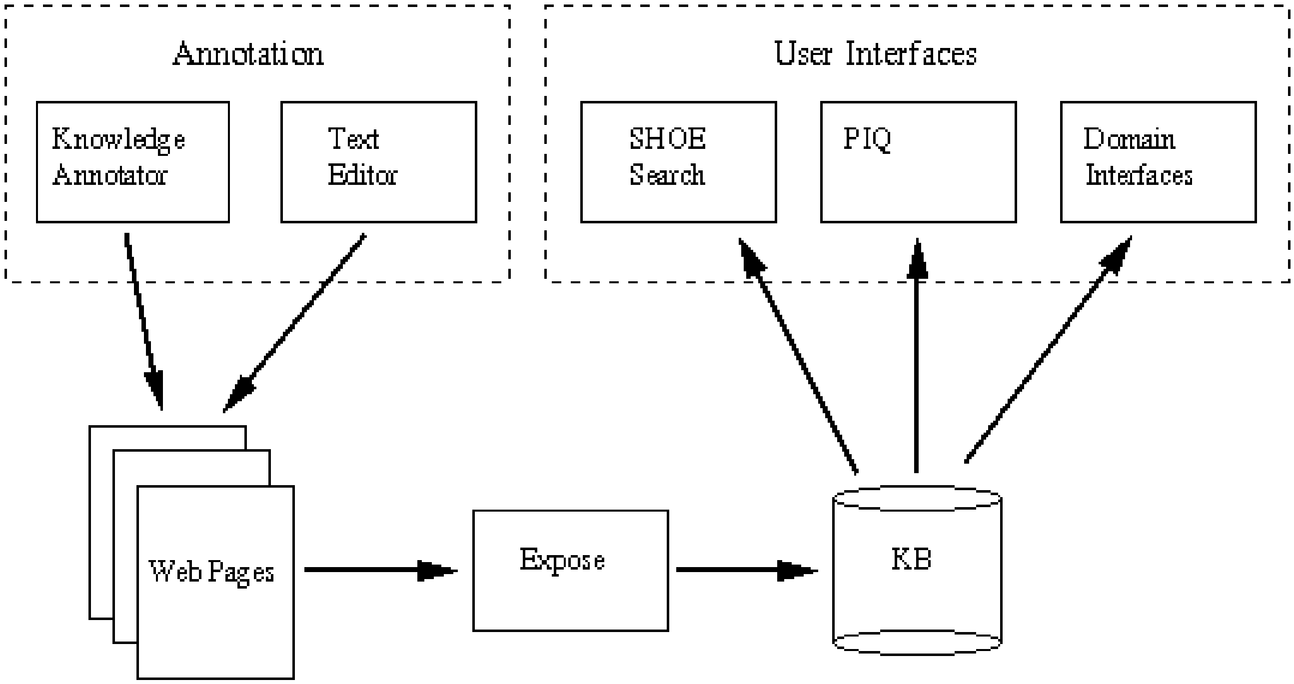

Figure 1. The System Architecture

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.

When SHOE pages are annotated and placed on the Web, they can be queried and indexed. Although it is possible to create agents that query pages in real time, we believe that it will be easier to parallel the way modern search engines work, that is 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 knowledge base (KB).

Currently, we store SHOE knowledge in a Parka KB (Stoffel, Taylor, and Hendler 1997), but all of our tools are designed with a generic KB API, so that an alternate knowledge representation (KR) 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 query efficiency and the most common types of inferences for SHOE. 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.

It is important to point out that in SHOE, two ontologies could use the same term to mean different

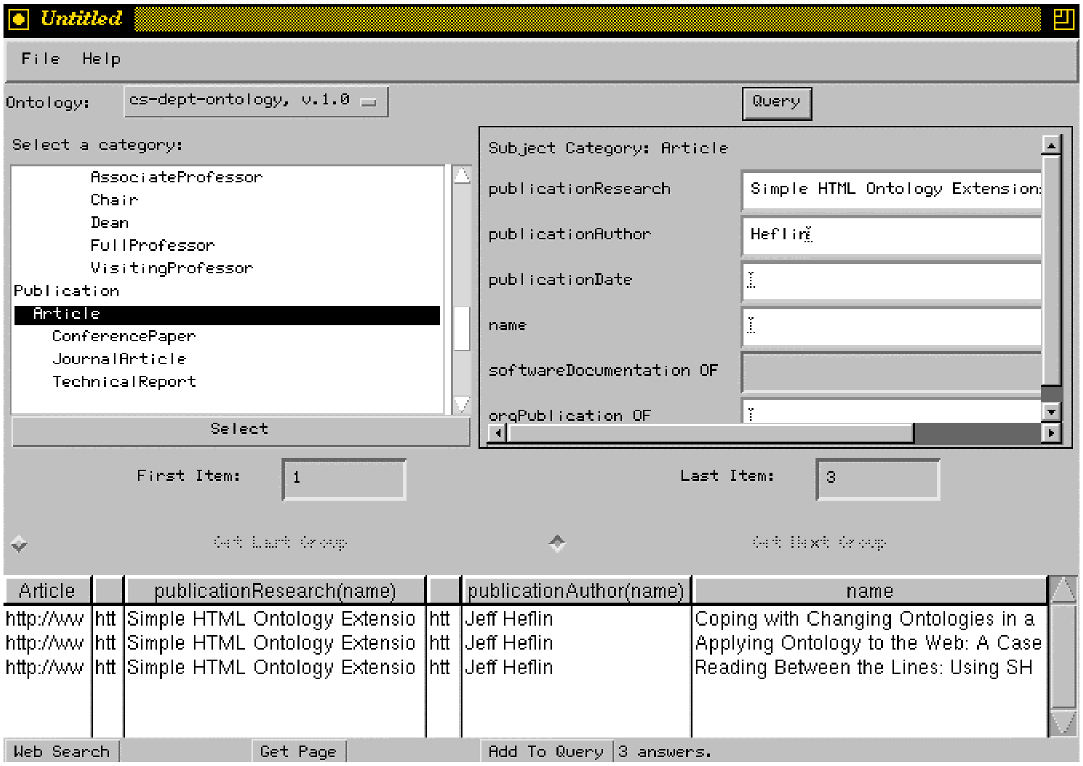

Figure 2. SHOE Search

things. This is necessary because ontologies can be designed by anyone, and there is no way to prevent terms from being used by multiple ontology authors. Therefore, when ontology terms are stored in the knowledge base, they are renamed by appending an ontology specific string.

Any number of query tools can access SHOE knowledge that has been loaded into a KB. Since the average web user does not want to take the time to learn complex KR languages,[1] we must provide query tools that are graphical in nature. Our first query tool was the Parka Interface for Queries (PIQ), which essentially allowed users to draw a graph in which nodes represented constant or variable instances and arcs represented relations. To answer the query, the system would perform subgraph matching on the user's graph. However, this was too complicated for all but the most advanced users and drawing queries was time consuming.

In order to support the application of SHOE to food safety (Heflin, Hendler, and Luke 1999), we developed the TSE Path Analyzer, a tool that allows a user to trace the possible pathways of food product contamination. The user only needs to select a few values from hierarchical lists to specify a query. The results are presented in the form of a graph, and clicking on any node in the graph will open the web page that provides details on that node. Note that because the Path Analyzer relies solely on SHOE information that has been gathered from web pages, it is possible for spatially distant communities with different levels of computational resources to contribute information.

The problem with the query tools described above is that the general-purpose tool is too complicated for the average user, while the tool that is easy to use is specific to a particular domain. This led us to try to find a balance: a general-purpose query tool that could used with a minimal learning curve.

3. SHOE Search

The idea behind the SHOE Search tool is that if queries are issued within a context, the tool can prompt the user with context specific information and can more accurately locate the information desired by the user. SHOE Search is written in Java, and can be launched from a web page via an applet. A screen shot of SHOE Search is shown in Figure 2.

The user selects a context by choosing an ontology from a drop-down list. The list of available ontologies are those that are known by the underlying KB. The identifiers and the version numbers of each ontology are displayed, so that users may choose to issue their queries against earlier versions of ontologies.

[1] Even the boolean searches offered as "advanced" features by contemporary search engines are too confusing for most users.

After the user chooses an ontology, the system creates a list of categories that are defined in that ontology. This list is organized so that specializations of categories are indented beneath them. This taxonomy makes it possible for the user to quickly determine the kinds of objects that are described in the ontology and to choose a class that is of sufficient granularity for his or her needs. Since one of the main purposes of choosing an ontology is to reduce the number of choices that the user will have to make subsequently, the list of categories generally does not include categories defined in ontologies extended by the selected ontology, even if they are ancestors of categories defined locally. It is assumed that if these categories are of interest to the user, then he can first select an appropriate ontology. However, ontologies may rename an element from an extended ontology and effectively "import" them. Such categories are included in the list, and displayed with their local name.

When the user chooses a category and presses the **Select** button, the system responds with a set of properties that are applicable for that category (in a frame-slot system, this would essentially be the slots of the selected frame). Applicable properties are inheritable; thus any properties that apply to an ancestor of the selected category are also included in the set. However, as with the list of available categories, it is important to provide some filtering for the user, so only those relations that are defined or aliased in the selected ontology will appear, even if other ontologies define relations that could be relevant.

Technically, SHOE does not define properties, instead it defines relations which can have some number of typed arguments. As such, a property of a class can be considered a relation where the first argument must be a member of that class. However, this makes the determination of properties dependent on the somewhat arbitrary ordering of arguments as chosen by the ontology designer. That is, the relation *workFor(Person, Organization)* would be a property of the class *Person*, but the inverse relation *hasEmployee(Organization, Person)* would be a property of *Organization*. In order to prevent SHOE Search queries from be restricted by these kinds of representational decisions, a relation in which the class is a subclass of the second argument is considered an inverse property and is included in the set available to the user. Such properties are clearly labeled in the display.

The property list allows the user to issue a query by example. He can type in values for one or more of the properties, and the system will only return those instances that match all of the specified values. Some of these property values are literals (i.e., strings, numbers, etc.) while others may be instances of classes. In the later case, it is unlikely that the user will know the keys for these instances, since these keys are typically URLs and the purpose of a search system is to locate URLs. Therefore, arbitrary strings are allowed in these fields and the query will attempt to match these strings to the names of instances. To increase the chance of a match, case-insensitivity and partial string matching are used.

When the users presses the **Query** button, the system constructs a conjunctive query and issues it to the KB. The first atom of the query specifies that the instance must be of the selected category, e.g., *everyInstanceOf(Person, ?K)*.[2] The remaining atoms depend on the type of the argument that the value represents. In the case of numbers, the atom is simply looking for an instance that has the specified value for the relation. In the case of strings, two atoms are added, one to find the values of the relation, and the other to perform a partial string match on them to the string specified by the user. Finally, if the type of the argument is a category, then three clauses are added: one to get the values for the relation, one to get the corresponding names of these instance keys, and a third to match the name strings to the string specified by the user. Note that even if the user only specified values for two properties, the resulting query could contain as many as seven conjuncts. One of the advantages of SHOE Search is that useful but complex queries are constructed automatically. For example, the query constructed by the user in Figure 2 corresponds to a Parka query of the form:

everyInstanceOf(Article, ?K) ∧
publicationResearch(?K, ?X1) ∧ *name(?X1, ?N1)* ∧
strMatch(?N1, "%Simple HTML Ontology Extensions%") ∧
publicationAuthor(?K, ?X2) ∧ *name(?X2, ?N2)* ∧
strMatch(?N2, "%Heflin%")

Many users would have difficulty constructing such queries by hand.

When the KB returns the results of the query, they are displayed in tabular format. The KB is likely to return many duplicate results; some of these will be due to redundancies of different web pages, others might be because the same page was visited many times using different URLs. Either way, duplicate results would simply clutter the display, and therefore they are removed before the system displays them. Generally, both the names and keys are displayed for related instances. In this way, the user can distinguish between instances that happen to have identical names. If the user clicks on an instance key, whether it is the instance that matches the query, or one that matches one of its properties, the corresponding web page is opened in a new browser window. This allows the user to browse the Web with more control over the queries.

Sometimes users may have trouble deciding what values to use for a given a property and may end up getting no results because incorrect values were entered. To remedy this problem, we have added a **Find** button next to each property that finds valid values for that

[2] In Parka, the *everyInstanceOf* predicate is used to return all instances of a category using the transitivity of category membership, as opposed to the *instanceOf* predicate which only returns those instances which have explicitly been asserted as a member of the category.

property. If this button is pressed the system will essentially issue a query to find all instances that have a value for the selected property and return those values in the tabular display. The user may then select one of these values and press the **Add To Query** button to insert it into the query field for the property. In order to do this, the system always keeps track of which columns of query results correspond to which properties.

The user may wish to view the values for a certain property without restricting the query. The **Show** checkbox allows the user to specify that an additional property should be displayed in the results. Note that properties for which the user has specified a value have this box checked by default. However, because the current system only supports conjunctive queries, this option can have unintuitive results. For example, if the user chooses to show a property for which no instances have a value, then no answers are returned, even if there are many possible answers for the rest of the query.

The **Show** checkbox and the **Add To Query** button can be used together to help the user gradually filter results and find the desired instances. The user starts by checking some of the **Show** boxes and issuing a query. One of the results can be selected and added to the query. When the query is reissued, fewer results should be returned. By repeating this process the user can continue to reduce the results returned to a manageable set.

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 have a **Web Search** feature that will translate the user's query into a similar search engine query and allow them to submit 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. For example the search engine query string generated for Figure 2 would be:

"Article" about research +"Simple HTML Ontology Extensions" by author +"Heflin"

The quality of results for these queries vary depending on the type of query and the search engine used. For search engines with advanced query capabilities, these queries could be expanded to included synonyms for terms using disjunction or positional information could be used to relate properties to their values.

One problem that SHOE Search queries have is that in their annotations users often specify relations between instances without explicitly declaring categories for these instances. Since the first conjunct of a SHOE Search query restricts answers to those instances that are members of a particular class, all such relations are ignored. However, we did not want to remove the classification conjunct because it is often useful, for example if the user specifies a query to find "All Journal Articles whose author is John Smith" then we do not want to return instances of conference papers by John Smith. Therefore, when a web page uses an instance in a relation and does not declare a category for it, the system assumes that the user implicitly means it is of the required type, and asserts the appropriate category declaration as well. This improves the number of relevant answers that SHOE Search can return, but may result in false classifications if the user did not understand the defined typing for the relation.

4. Related Work

There are numerous efforts to create semantic languages for the Web. The Ontobroker project (Fensel et al. 1998) uses a language to describe data that is embedded in HTML, but relies on a centralized broker for ontology definitions. Tool support for Ontobroker includes a hyperbolic view for exploring ontologies, a simple text-based interface for specifying queries in frame logic, and an advanced interface that helps the user build frame logic queries using pull-down lists. The Ontology Markup Language (OML) and Conceptual Knowledge Markup Language (CKML) (Kent 1999) are used together for semantic markup that is based on the theories of formal concept analysis and information flow. The W3C has developed the Resource Description Framework (RDF) (Lassila and Swick 1999), which uses XML to specify semantic networks of information on web pages but has only a weak notion of ontologies. The RDF Schema proposal (Brickley and Guha 1999) improves this situation somewhat, but does not sufficiently handle the notions of revising or integrating ontologies.

5. Conclusion

We have described a process for using SHOE to improve search on the Web and have presented tools that demonstrate these capabilities. It is important to note that this system parallels the way the Web works today: a standard language allows a diverse tool set to interact indirectly. Markup can be performed using a text editor, the Knowledge Annotator, or other custom-built annotation tools. Any number of web-crawlers or agents can be built that extract SHOE content from web pages and this content can be stored and queried in different ways. Thus, SHOE-enabled search engines could differentiate themselves by coverage, inferential capability, speed and user interfaces.

The SHOE Search tool is unique in that it allows the user to specify a context for search, and then provides the user with options that are relevant to that context. It is essentially a frame-based query by example interface, but includes features that allow the user to discover the content of the knowledge base and to extend the search beyond the knowledge base by translating the query into a format that can be issued to standard search engines. Since the user has been prompted to enter values for defining characteristics of the object in question, search engine queries created by SHOE Search are more likely to return relevant results.

The biggest barrier to the SHOE solution is the knowledge acquisition problem. However, adding SHOE annotations to web pages is only moderately more time consuming than converting them to standard XML. We feel that if users can be convinced of the benefits of semantic markup, then they would be more willing to take the time to do it. Nevertheless, automatic and semi-automatic solutions will be necessary to achieve a critical mass. Therefore we are examining approaches to extract SHOE from semi-structured web pages, to translate documents that use common XML DTDs to SHOE, and to translate other semantic web languages such as RDF to SHOE.

Knowledge representation tools for the Web must be geared toward the average user, who often does not have the time or desire to learn first-order logic. The suite of SHOE tools, particularly SHOE Search, are a step in this direction, but there is much room for improvement. Most of these tools are available as on-line demos, and we encourage the interested reader to visit our website at http://www.cs.umd.edu/projects/plus/SHOE/ and provide feedback. We believe that in the future, languages and tools such as those we have developed for SHOE will be crucial in locating relevant information in the ever expanding and changing Web.

Acknowledgments

This work was supported by the Army Research Laboratory under contract number DAAL01-97-K0135. We would like to thank Carolyn Gasarch who contributed to the development of the SHOE Search tool. Professor Hendler is currently working as a Program Manager for the Defense Advanced Research Projects Agency (DARPA). 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.

References

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

Fensel, D., Decker, S., Erdmann, M., and Studer, R. 1998. 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.

Heflin, J., and Hendler, J. 2000. Dynamic Ontologies on the Web. In *Proceedings of American Association for Artificial Intelligence Conference (AAAI-2000)*. Menlo Park, Calif.: AAAI Press.

Heflin, J., Hendler, J., and Luke, S. 1999. Applying Ontology to the Web: A Case Study. In: J. Mira, J. Sanchez-Andres (Eds.), *International Work-Conference on Artificial and Natural Neural Networks, IWANN'99. Proceedings, Volume II*. 715-724. Berlin: Springer.

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

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

Luke, S. and J. Heflin. 1997. *SHOE 1.0, Proposed Specification*. At http://www.cs.umd.edu/projects/plus/SHOE/spec.html

Stoffel K., Taylor, M., and Hendler, J. 1997. Efficient Management of Very Large Ontologies. In *Proceedings of American Association for Artificial Intelligence Conference (AAAI-97)*. Menlo Park, Calif.: AAAI Press.