Coping with Changing Ontologies in a Distributed Environment

Jeff Heflin, James Hendler, and Sean Luke

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

Abstract

We discuss the problems associated with versioning ontologies in distributed environments. This is an important issue because ontologies can be of great use in structuring and querying internet information, but many of the Internet's characteristics, such as distributed ownership, rapid evolution, and heterogeneity, make ontology management difficult. We present SHOE, a web-based knowledge representation language, that supports multiple versions of ontologies. We then discuss the features of SHOE that address ontology versioning, the affects of ontology revision on SHOE web pages, and methods for implementing ontology integration using SHOE's extension and version mechanisms.

1. Introduction

As the use of ontologies becomes more prevalent, there is a more pressing need for good ontology management schemes. This is especially true once an ontology has been used to structure data, since changing it can be very expensive. Often the solution is to "get it right the first time", however, in long term applications, there is always the chance that new information will be discovered or that different features of the domain will become important. Therefore, we must think of ontology development as an ongoing process. In a centralized environment, it may be possible to coordinate ontology revisions with corresponding revisions to the data that was structured using the ontology. However, as the volume of data increases this become more difficult. In distributed environments where ontologies are used for interoperation, this problem is compounded by the fact that the data is likely to be owned by various parties and spread across servers. As such, some parties may not be prepared for a revision and other may be opposed to it.

Ontologies can be of great use in structuring and querying internet information, but because the Internet is so dynamic and distributed, managing the inevitable changes in ontologies will be extremely difficult. We have considered these problems in our work on Simple HTML Ontology Extensions (SHOE), a language that is embedded in web pages. In this paper, we describe the features of SHOE that address the problem of evolving ontologies, describe how ontology revision affects SHOE web pages, and discuss how ontology integration can be accomplished using SHOE's extension and version mechanisms.

2. SHOE

The underlying philosophy of SHOE is that intelligent internet agents will be able to better perform their tasks if the most useful information is provided in a structured manner. To this end, SHOE extends HTML with a set of knowledge oriented tags that, unlike HTML tags, provide structure for knowledge acquisition as opposed to information presentation. In addition to providing explicit knowledge, SHOE sanctions the discovery of implicit knowledge through the use of taxonomies and inference rules available in reusable ontologies that are referenced by SHOE web pages. This allows information providers to encode only the necessary information on their web pages, and to use the level of detail that is appropriate to the context. SHOE-enabled web tools can then process this information in novel ways to provide more intelligent access to the information.

To achieve compatibility with existing web standards, SHOE's syntax is defined as an application of SGML, a language that defines tag-based languages and was the influence for HTML's syntax. SHOE is also compatible with XML[1], an emerging standard for the Internet that is a restricted form of SGML.

2.1 Language Features

In this section we describe the features of SHOE that are necessary for an understanding of this paper. A complete description can be found in (Luke and Heflin 1997).

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. 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. Ontologies inherit all of the components present in their ancestors. An example ontology is shown in Figure 1.

Ontology reuse in SHOE is accomplished by extending general ontologies to create more specific ontologies. Specifically, the <USE-ONTOLOGY> tag indicates the id and version number of an ontology that is extended. A

[1] A minor variation of the syntax must be used for a SHOE document to be a well-formed XML document.

```
<HTML>
...
<BODY>
<ONTOLOGY ID="cs-dept-ontology" VERSION="1.1"
          BACKWARD-COMPATIBLE-WITH=”1.0”>
<USE-ONTOLOGY ID="univ-ontology" VERSION="1.0" PREFIX="u"
              URL=”http://ontlib.org/univ_v1.0.html”>
...
<DEF-RENAME FROM=”u.Department” TO=”Department”>
<DEF-RENAME FROM=”u.org.WorksFor” TO=”facultyOf”>
...
<DEF-CATEGORY NAME="Artificial_Intelligence" ISA="u.Research_Area">
...
<DEF-RELATION NAME="softwareLangauge">
  <DEF-ARG POS=1 TYPE="u.Research_Project">
  <DEF-ARG POS=2 TYPE="Computer_Language">
</DEF-RELATION>
...
</ONTOLOGY>
</BODY>
</HTML>
```

Figure 1. An Example Ontology

URL field allows systems to locate the ontology if needed and a PREFIX field 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 in a general ontology.

An ontology can define categories, relations, and other components. Categories are defined with a <DEF-CATEGORY> tag and may specify one or more subsuming categories. Relations, which are essentially n-ary predicates, are defined with a <DEF-RELATION> tag and must specify types for each argument.

Sometimes an ontology may wish to use a term from another ontology, but a different label may be more specific 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, even though the concept has been standardized across domains.

SHOE uses inference rules, indicated by the <DEF-INFERENCE> tag, to supply additional axioms. These rules are similar to Horn clauses in that negations are not allowed in implications[2], thus computational complexity is reduced. A SHOE inference rule consists of a body of one or more subclauses describing claims that entities might make, and a head 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 head and body 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. The following example illustrates the construction of an inference that uses all three types of subclauses.

```
<DEF-INFERENCE>
<INF-IF>
   <CATEGORY NAME=”Car” VAR FOR=”X”>
   <RELATION NAME=”age”>
      <ARG POS=1 VAR VALUE=”X”>
      <ARG POS=2 VAR VALUE=”A”>
   </RELATION>
   <COMPARISON OP=”greaterThan”>
      <ARG POS=1 VAR VALUE=”A”>
      <ARG POS=2 VALUE=”25”>
   </COMPARISON>
</INF-IF>
<INF-THEN>
   <CATEGORY NAME=”Antique” VAR FOR=”X”>
</INF-THEN>
</DEF-INFERENCE>
```

The data sources for SHOE are web pages, where each can be thought of as a miniature knowledge base. These 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> tag that has a field 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

[2] In conjunctive normal form a Horn clause can have no more than one positive literal.

guaranteed to identify only a single resource. An instance states that it is based on an ontology with the <USE-ONTOLOGY> tag, which has the same function as the identically named tag used within ontologies. The use of common ontologies makes it possible to issue a single logical query to a set of data source and enables the integration of related domains. To prevent ambiguity in the declarations, ontology elements are referred to using the prefixing mechanism described earlier.

2.2 Mapping SHOE to First Order Logic

In order to analyze certain aspects of SHOE, we will map it into a first order logical theory. This mapping will intentionally omit some features of the language so that we may focus on the problem of ontology revision.

Essentially an ontology can be thought of as a tuple <V,A> where V is the vocabulary and A is the set of axioms[3] that govern the theory. In first-order logic terms, V is the set of non-logical symbols and A is a set of definite program clauses. For convenience, we will assume that the symbols in the vocabulary of each ontology are distinct. In actuality, SHOE has a separate namespace for each ontology, but we can make our assumption without loss of generality since it is possible to apply a renaming that appends a unique ontology identifier to each symbol. We now discuss how to build V and A, based upon the components that are defined in the ontology:

A <USE-ONTOLOGY> statement adds the vocabulary and axioms of the specified ontology to the current ontology. Since we have assumed that names must be unique, we do not have to concern ourselves with name conflicts.

A <DEF-RELATION> adds a symbol to the vocabulary and for each argument type that is a category, adds an axiom that states that an instance in that argument must be a member of the category. If the tag specifies a name R and has n arguments then there is an n-ary predicate symbol R in V. If the type of the i^{th} argument is C, then $[R(x_1,...,x_i,...x_n) \rightarrow C(x_i)] \in A$

A <DEF-CATEGORY> adds a unary predicate symbol to the vocabulary and possibly a set of rules indicating membership. If the name is C, then $C \in V$. For each super-category P_i specified, $[C(x) \rightarrow P_i(x)] \in A$.

A <DEF-INFERENCE> adds one or more axioms to the theory. If there is a single clause in the <INF-THEN>, then there is one axiom with a conjunction of the <INF-IF> clauses as the antecedent and the <INF-THEN> clause as the consequent. If there are n clauses in the <INF-THEN> then there are n axioms, each of which has one of the clauses as the consequent and has the same antecedent as above.

A <DEF-RENAME> provides an alias for a non-logical symbol. It is meant as a convenience for users and can be implemented using a simple pre-processing step that translates the alias to the original, unique non-logical symbol. Therefore, it can be ignored for the logical theory.

A data source, such as a knowledge base or intelligent agent, uses one or more ontologies to make relation and category claims. If we have a data source S, let V_S be the union of the vocabulary of these ontologies and let A_S be the union of the axioms. We say S is well-formed if each category and relation claim is a ground atom of the form $p(t_1,...,t_n)$ where p is a predicate symbol such that $p \in V_S$. If D_S is the set of ground atoms, then we can define interpretations and models of S using their usual definitions when applied to $D_S \cup A_S$.

A SHOE query Q=<G,O,S> consists of a goal G, an ontology O, and a data source S. Here, S does not have to be a single web page, it could be a collection of web pages, possibly even the entire web. G is well-formed with respect to O=<V,A> if it is well-formed under the standard logical definition given V provides the non-logical symbols. Q is well formed if both S is well-formed and G is well-formed with respect to O. Note that by changing the ontology O and leaving G and S constant, it is possible to achieve different answers to the same question, since different ontologies could result in a different set of inferences (via category membership or other axioms). In this case, we say that each ontology provides a different *query perspective* for the data source; it changes the possible outcome of queries issued against the source.

We construct a first order logic program P from the union of axioms of O and the ground atoms in S that are well-formed with respect to O. Note that unless $V \supseteq V_S$, it is possible that there are some atoms that will not be formed; therefore we discard them as being irrelevant to the query. We can now define an answer in the usual logic programming way: an answer for $P \cup \{G\}$ is a substitution for variables of G. We then say θ is a correct answer for $P \cup \{G\}$ if $G\theta$ is ground and is a logical consequence of P. If Q is not well-formed then there are no answers to the query.

3. Revisioning

An ontology revision is a change in the components of an ontology. Thus, it can involve the addition or removal of categories, relations, and/or axioms. A revision may also extend a new ontology or stop extending one.

3.1 Effects of Revisions

We describe revisions by how they change an ontology. For each type of revision, we discuss its relevance to data sources and queries. This discussion only describes revisions that add or remove components; the modification of a component can be thought of as a removal followed by an addition.

If a revision O′ consists of the removal of categories or relations from O, then there exists a query Q=<G,O,S> such that Q′=<G,O′,S> cannot be asked or will return fewer answers than Q. This is because G or some atoms in S that were well-formed w.r.t. O will not be well-formed w.r.t. O′. Informally, if categories or relations are removed,

[3] We distinguish axioms from rules by using the former term in the general sense and the later term to refer specifically to SHOE inference rules.

predicate symbols are removed from the vocabulary. There exist sentences that depend on these symbols for well-formedness; when the symbols are removed the sentences are no longer well-formed. If these sentences are the goal G then Q′ is not well-formed, if they are ground atoms of the data source S, then the new program $P' \subseteq P$. Revisions of this type may mean that some legacy data sources cannot be queried with the new ontology and others may not be queried in their entirety.

If a revision O′ adds an arbitrary rule, then the new query will still be well-formed since the vocabulary is identical to that of the original ontology. Additionally, if θ is a correct answer for Q=<G,O,S> then θ is a correct answer for Q′=<G,O′,S>. Essentially this is because the revision only adds clauses to the logic program and first-order logic is monotonic. Thus, a revision which adds rules can always be used to query legacy data sources, but there may be additional answers that were not originally intended by the author of the data. Similar reasoning is used to ascertain that if the revision removes rules, then fewer query answers may be retrieved.

Finally, if the revision adds categories or relations, then new queries are still well-formed since the new vocabulary includes all of the predicate symbols that were in the old vocabulary. Additionally, θ is a correct answer for query Q=<G,O,S> iff θ is a correct answer for Q′=<G,O′,S>. The forward direction is trivial due to the monotonicity of first order logic. The reverse direction depends on the nature of the axioms added: $R(x_1,...,x_i,...x_n) \rightarrow C(x_i)$ for relations and $C(x) \rightarrow P_i(x)$ for categories. Due to the definitions of categories and relations, the predicate of the antecedents is a symbol added by the new ontology and must be distinct from symbols in any other ontology. Therefore any atoms formed from these predicates are not well-formed with respect to any other ontology. Therefore, there can be no such atoms in S, since S must be well-formed with respect to some ontology ≠ O′. Since the antecedents cannot be fulfilled, the rules will have no new logical consequences that are ground atoms. Therefore, any answer that is correct for Q=<G,O,S> must also be correct for Q=<G,O′,S>. This result indicates that we can safely add relations or categories to the revision, and expect legacy data sources to be unaffected by the change. In other words, any queries against a data source that uses the revised ontology as a basis will have exactly the same results as queries that use the original as a basis.

3.2 Versioning

As described in the previous section, there are some revisions that cause data sources to be ill-formed and there are others which cause queries issued against them to have different results. The first case results in the loss of access to useful data, the later in potentially unexpected behavior. It is these situations that makes SHOE's versioning scheme important. SHOE maintains each version of the ontology and an instance must state which version it is referencing. Thus, data sources can upgrade to the new ontology at their own pace and some may never upgrade.

To accomplish a revision in SHOE, the ontology designer copies the original ontology file, assigns it a new version number, and adds or removes elements as needed. If the revision merely adds ontology elements, then it does not require any changes in existing web pages to be used in interpreting them. Therefore, it can specify that it is compatible with previous versions using the optional BACKWARD-COMPATIBLE-WITH field in the <ONTOLOGY> tag. Agents and query systems that discover this ontology can also use it in place of any of the ontologies that it is backward-compatible with. However, there is a danger in this as we describe below.

Consider the following scenario: an individual for whatever reasons, malicious or otherwise, decides that they want to create a revision to a popular ontology that is owned by somebody else. This revision only adds components, and thus is compatible with all existing web pages that reference the original ontology. The individual then indicates that this ontology is backward-compatible with the original. Unless there is a mechanism to determine if a revision is official, any agents or query systems that come across the revision will assume that they can use it in place of the old one, and unintended inferences may result. We suggest three methods to prevent this:

- agents will only use the revision as a substitute if it only adds categories or relations. Since such revisions do not change the results of queries issued against existing data sources, it does not matter if it is an official revision.
- a revision must be located on the same server and in the same path as the ontology it revises. This guarantees that the owner has made the revision, but makes it difficult to move the location of an ontology once it has been used.
- the original ontology must authorize the revision. This could be accomplished by a <REVISED-BY> tag that points to the location of the revision. To use this method, upon discovering a purported revision, a system should reload the original ontology and see if it authorizes the revision.

Currently, we recommend that SHOE systems use the second approach, although we are considering the inclusion of a revised-by tag in a future version of the language.

4. Implementing Ontology Integration

The preferred method of ontology development in SHOE is to extend existing ontologies and create new definitions only when existing definitions are unsuitable. In this way, all concepts are automatically integrated. However, when there is concurrent development of ontologies in a large, distributed environment such as the Web, it is inevitable that new concepts will be defined when existing ones could be used. In these situations, manual integration of ontologies may eventually be needed.

Ontology integration (Jannink et al. 1998; Grüninger 1996) typically involves identifying the correspondences between two ontologies, determining the differences in definitions, and creating a new ontology that resolves these differences. Wiederhold (1994) describes four types of domain differences, which we paraphrase here:

- terminology: different names are used for the same concepts
- scope: similar categories may not match exactly; their extensions intersect, but each may have instances that cannot be classified under the other
- encoding: the valid values for a property can be different, even different scales could be used
- context: a term in one domain has a completely different meaning in another

When ontologies have been implemented, then simply creating a new integrated ontology does not solve the problem of integrating information, since all of the data sources would have to be revised to reflect the new ontology. We suggest three ways to implement ontology integration.

In the first approach, we create a mapping ontology that extends both ontologies and assign it a unique id that distinguishes it from all other ontologies. We then add a <USE-ONTOLOGY> tag for each ontology to be integrated. Since this ontology has access to objects from both domains, it can create inference rules using the <DEF-INFERENCE> tag to map the common items between them. Terminological differences can be mapped using simple if-and-only-if rules. For example:

$$\text{BusOnt1.Employee}(x) \Leftrightarrow \text{BusOnt2.StaffMember}(x)$$

Note that since SHOE's rules only allow inference in one direction, if-and-only-if rules like this one are actually implemented as two rules, one for each direction. Scope differences require mapping a category to the most specific category in the other domain that subsumes it. For example:

$$\text{AF\_Ont.FighterPilot}(x) \Rightarrow \text{FAA\_Ont.JetPilot}(x)$$

Encoding differences require rules that map individual values as in:

$$\text{CriticOnt1.Rating}(x,\text{"Good"}) \Leftrightarrow \text{CriticOnt2.Rating}(x,\text{"3"})$$

Finally, context differences can be handled by automatically assuming that terms never match unless explicitly instructed as suggested by (Wiederhold 1994). The advantage of a mapping ontology is that the domain ontologies are unchanged, thus it can be created by anyone as needed. The disadvantage is that the mapping of many domains could result in a complex mess of mapping ontologies.

Another approach to implementing integration is to revise each ontology to include mappings to the other. First, we create a new version of each ontology, called a mapping revision, and assign it an appropriate version number. To each revision, we add mapping rules similar to the ones described in the mapping ontology approach. Since we are only adding inference rules, we can use the BACKWARD-COMPATIBLE-WITH field to specify that the revisions can be used in place of the original ontologies. If these revisions are then used as the query perspectives for data sources based on the domain ontologies, then data integration occurs without modifying the data sources.

The disadvantage of the two mapping approaches is that they ignore a fundamental problem: the overlapping concepts do not belong in either domain, they are more general. The fact that two domains share the concept may mean that other domains will use it as well. If this is so, then each new domain would need a set of rules to map it to the others. Obviously this can become unwieldy very quickly. A more natural approach is to merge the common items into a more general ontology, called an intersection ontology, which is then extended by revisions to the domain ontologies. We create a new ontology to serve as the intersection ontology and give it a unique identifier. Then we standardize the common elements from the source ontologies and add them to the intersection ontology. To create the revisions for each source ontology, we create new versions with appropriate version numbers and add the <USE-ONTOLOGY> tag to each revision to specify that it uses the intersection ontology. Since the intersection ontology must use a standard terminology, existing data sources will only be well-formed with respect to the new ontologies if we translate the standard terminology to the domain terminology. This is done by replacing the domain definitions of the common elements with <DEF-RENAME> tags that rename the intersection ontology's standardized names to the names originally used by the source ontology. This has the advantage that equivalence of terms can be determined in the preprocessing phase rather than at query execution time. Finally, we use the BACKWARD-COMPATIBLE-WITH field to indicate that these revisions are compatible with the original ontologies.

5. Related Work

In the last decade, there has been much active research in the area of ontology. For an overview and comparison of ontology design projects, see (Noy and Hafner 1997). The theme of ontology reuse has been accepted as an important one, and many approaches have been suggested. One of the most widely used notions is, as we have done with SHOE, to use general ontologies as the foundation for building more specific ontologies. Ontolingua (Farquhar et al. 1997) allows ontologies to include other ontologies and even allows cyclic inclusion. In (Borst et al. 1996), Ontolingua is used to perform step by step addition of new ontological distinctions. Instead of importing ontologies, (Grüninger 1996) describes the construction of ontologies from building blocks that are classes of sentences in some foundational theory. In contrast to the top-down approach of building ontologies from general components, Jannink et al. (1998) suggest that an ontology should be built for each data source, and generalization is accomplished by integrating these data sources. In this way, the data dictates the structure of the ontology rather than the other way around.

Most ontology work does not consider ontology development in a distributed environment. One exception is the Ontolingua Server (Farquhar et al. 1997) which can be used by many to collaborate on ontology development.

However, the key difference between the Ontolingua Server and the work discussed in this paper is that the former only deals with managing the development of ontologies, while the later discusses features needed for distributed ontology implementation.

Other projects are using ontologies to provide some structure to Web, but none of these have focused on the problem of maintaining consistency as the ontologies evolve. The Ontobroker (Fensel et al. 1998) project uses a language that, like SHOE, is embedded in HTML. Although the syntax of this language is more compact, it is not as easy to understand as SHOE. Also, Ontobroker does not have a mechanism for pages to use multiple ontologies and those who are not members of the community have no way of discovering the ontology information. The Web Analysis and Visualization Environment (WAVE) project (Kent and Neuss 1995) has designed the Ontology Markup Language (OML) and Conceptual Knowledge Markup Language (CKML), both of which are based on early versions of SHOE. The W3C is developing the Resource Description Framework (RDF) (Lassila and Swick 1998). RDF uses XML to specify semantic networks of information on web pages, but has only a weak notion of ontologies. Since there is no controlled vocabulary, integration must be performed pair-wise on data sources. This is complicated by the fact that there is no way to specify reusable mapping rules.

6. Conclusions

We have discussed the problems associated with managing ontologies in a dynamic, distributed, and heterogeneous environment such as the Web. We have described the components of a SHOE ontology and focused on those that allow revisioning. We have mapped SHOE into a first order logic definite program and used this to discuss how different types of revisions to an ontology affect existing data sources. We have shown that revisions that add categories or relations will have no effect, revisions that modify rules may change the answers to queries against the data sources, and revisions that remove categories or relations may make the ontology incompatible with the data sources. This knowledge should be used in weighing the benefits and costs of any revision. Although ideally integration is a byproduct of ontology extension, in a large, distributed environment manual ontology integration will need to be performed periodically. We described three methods of implementing integration, showed how they could be applied in SHOE, and discussed the tradeoffs of each.

The Web is currently an untamed wilderness, but ontologies can be used to give it a sense of order. However, we cannot expect the Web to stop growing and changing simply because we have fit it into the framework of an ontology. Instead, the ontologies must evolve with the Web, and must allow for different growth rates in different areas. We believe that SHOE accomplishes these objectives and makes internet interoperability possible.

Acknowledgments

This work was supported by the Army Research Laboratory under contract number DAAL01-97-K0135.

References

Borst, P.; Benjaminm, J.; Wielinga, B.; and Akkermans, H. 1996. An Application of Ontology Construction. In *Proc. of ECAI'96 Workshop on Ontological Engineering*.

Farquhar, A., R. Fikes, and J. Rice. 1997. Tools for Assembling Modular Ontologies in Ontolingua. In *Proc. of AAAI-97*, 436-441. Menlo Park, CA: AAAI Press.

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

Gruber, T. 1993. A Translation Approach to Portable Ontology Specifications, Technical Report, KSL 92-71, Dept. of Computer Science, Stanford University.

Grüninger, M. 1996. Designing and Evaluating Generic Ontologies. In *Proc. of ECAI'96 Workshop on Ontological Engineering*.

Jannink, J.; Pichai, S.; Verheijen, D.; and Wiederhold, G. 1998. Encapsulation and Composition of Ontologies. In *AI and Information Integration, Technical Report WS-98-14*, 43-50. Menlo Park, CA: AAAI Press.

Kent, R.E. and C. Neuss. 1995. Creating a Web Analysis and Visualization Environment. *Computer Networks and ISDN Systems*, 28.

Lassila, O. and R.R. Swick. 1998. *Resource Description Framework (RDF) Model and Syntax*. W3C (World-Wide Web Consortium). At http://www.w3.org/TR/WD-rdf-syntax-19980216.html.

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

Noy, N.; and Hafner, C. 1997. The State of the Art in Ontology Design. *AI Magazine* 18(3):53-74.

Wiederhold, G. 1994. An Algebra for Ontology Composition. In *Proc. of 1994 Monterey Workshop on Formal Methods*, 56-62. U.S. Naval Postgraduate School.