Dynamic Ontologies on the Web

Jeff Heflin and James Hendler

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

Abstract

We discuss the problems associated with managing ontologies in distributed environments such as the Web. The Web poses unique problems for the use of ontologies because of the rapid evolution and autonomy of web sites. We present SHOE, a web-based knowledge representation language that supports multiple versions of ontologies. We describe SHOE in the terms of a logic that separates data from ontologies and allows ontologies to provide different perspectives on the data. We then discuss the features of SHOE that address ontology versioning, the effects of ontology revision on SHOE web pages, and methods for implementing ontology integration using SHOE's extension and version mechanisms.

1. Introduction

The World Wide Web is a repository of information that is structured for presentation to human readers and is thus mostly inaccessible to machines. This situation will be somewhat alleviated by the Extensible Markup Language (XML), which allows content to be separated from presentation. However, although XML Document Type Declarations (DTDs) can specify the grammar of markup languages, there are no facilities for formalizing the meaning of these languages. To create a web language with semantics, one must extend XML with features of knowledge representation (KR) languages.

However, the Web presents new challenges for KR; simply creating an XML syntax for traditional KR languages is insufficient. The Web is a distributed system and there are many providers of information. As such, the reliability of information is questionable, and it is inevitable that inconsistencies will arise. The Web is also in a constant state of change. Although standard vocabularies will be necessary for interoperability, these vocabularies must be able to evolve as the Web does. Additionally, the sheer size of the Web will test the scalability of KR systems and algorithms. As a consequence of the Web's size and dynamic nature, it must be treated as an open-world, since it would be unrealistic for any agent to assume that it knows all true facts about the Web.

The Simple HTML Ontology Extensions (SHOE) is an ontology-based knowledge representation language that is embedded in web pages (Luke et al. 1997; Heflin, Hendler, and Luke 1999). Over the course of four years, we have investigated the use of ontologies to support the structuring and querying of data on the Web. We begin this paper with an overview of the SHOE language, and then provide a logical semantics for it. We then discuss the problem of ontology revision, which is necessary in a dynamic environment such as the Web, and describe how SHOE's versioning mechanism copes with this. Next, we discuss the tendency of distributed ontologies to diverge, and provide methods for reintegrating them using SHOE's extension and version mechanisms. Finally, we discuss related work and present our conclusions.

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. SHOE associates meaning with this content by making each web page commit to one or more ontologies. These ontologies permit the discovery of implicit knowledge through the use of taxonomies and inference rules, allowing 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. Interoperability is promoted through the sharing and reuse of ontologies.

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. A slight variant of the syntax exists for compatibility with XML, and can be used by web sites that have begun to migrate to XML.

The nature of the SHOE language makes it possible to develop numerous tools and architectures for processing it. In order to evaluate the language, we have built a suite

```
<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-CATEGORY NAME="ComputerScience" ISA="u.ResearchArea">
...
<DEF-RELATION NAME="writtenIn">
  <DEF-ARG POS=1 TYPE="Program">
  <DEF-ARG POS=2 TYPE="ComputerLanguage">
</DEF-RELATION>
...
<DEF-RENAME FROM="u.Department" TO="Department">
<DEF-RENAME FROM="u.Chair" TO="DepartmentHead">
...
</ONTOLOGY>
</BODY>
</HTML>
```

Figure 1. An Example Ontology

of tools, including a tool for adding SHOE markup to web pages, a web crawler that gathers SHOE from web pages and stores it in a knowledge base, and a number of query tools. We have applied these tools to various domains, including computer science departments and food safety. A discussion of these tools and applications can be found in Heflin, Hendler and Luke (1999). Demos of the tools are available at *http://www.cs.umd.edu/projects/plus/SHOE*.

2.1 Language Features

In this section we describe the features of SHOE that are necessary for an understanding of this paper. The reader may be surprised by the simplicity of the language. In the context of the Web this is actually a virtue. In order for a language to be accepted by the web community, it must be easy for users to understand and tool developers to implement. As such, a guiding principle of the SHOE project has been to add features to the language only when it became clear that they were necessary.

SHOE ontologies are made publicly available by locating them on web pages. An example ontology is presented in Figure 1. An <ONTOLOGY> tag delimits the machine-readable portion of the ontology and specifies a unique identifier and a version number for the ontology. As is common in many ontology efforts, such as Ontolingua (Farquhar, Fikes, and Rice 1997) or Cyc (Lenat and Guha 1990), 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. As a result, ontologies are interoperable to the extent that they share the same ancestor ontologies.

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. An optional 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 introduced with a <DEF-CATEGORY> tag and may specify one or more subsuming categories. Note that it is not possible to specify subsuming categories for a category defined in another ontology. 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 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 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. SHOE rules can be reduced to Horn clauses, which allows implementations to take advantage of algorithms developed for datalog programs.

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, which states "If X is a Car and the age of X is greater than 25, then X is an Antique," 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 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 tag used within ontologies. 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. To prevent ambiguity in the declarations, ontology elements are referred to using the prefixing mechanism described earlier.

It is important to note that the features of the language have been carefully chosen to prevent logical inconsistency. There are no negations, no single-valued relations, and no disjoint constraints between categories. Although this may seem extremely restrictive, it is very useful for practical systems on the Web, where inconsistencies would otherwise be unavoidable.

2.2 Mapping SHOE to First Order Logic

In order to provide formal semantics for 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 issue of dynamic, distributed ontologies.

Essentially an ontology O can be thought of as a tuple <V,A> where V is the vocabulary and A is the set of axioms[1] that govern the theory. In the terminology of first-order theory, V is the set of predicate symbols, each with some arity >= 0, and A is a set of Horn 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 a category C, then $[R(x_1,...,x_i,...x_n) \rightarrow C(x_i)] \in A$. Note that this rule is necessary because the Web must be treated as an open-world. Since there is no way to know that a given object is *not* a member of the required category, it is better to assume that this information is as yet undiscovered than to assume that the relation is in error.

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 formula F is well-formed with respect to O if: 1) F is an atom of the form $p(t_1,...,t_n)$ where p is a n-ary predicate symbol such that $p \in V$ or 2) F is a Horn clause where each

[1] 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.

atom is of such a form. An ontology is well-formed if every axiom in the ontology is well-formed with respect to the ontology.

A data source, such as a knowledge base or intelligent agent, uses an ontology to make relation and category claims. Let $S=<O_S,D_S>$ be such a data source, where $O_S =<V_S,A_S>$ is the ontology and D_S is the set of claims. We say S is well-formed if O_S is well-formed and each element of D_S is a ground atom that is well-formed with respect to O_S. The terms of these ground atoms are constants and can be instance keys or values of a SHOE data type.

We introduce a perspective $P=<S,O>$ as a data source $S=<O_S,D_S>$ viewed in the context of an ontology $O=<V,A>$. If $O=O_S$ then we say that P is the intended perspective, otherwise it is an alternate perspective. It is possible that elements of D_S will not be well-formed with respect to O, such elements are considered to be irrelevant to the perspective. If we let W_S be the subset of D_S that is well-formed with respect to O, then we say that P results in a theory $T=W_S \cup A$. An interpretation of the perspective consists of a domain, the assignment of each constant in S to an element of the domain, and an assignment of each element in V to a relation from the domain. A model of P is an interpretation such that every formula in its theory T is true with respect to it. If every ground logical consequence[2] of perspective P is also a ground logical consequence of perspective P′ then we say that P′ semantically subsumes P. That is, any query issued against perspective P′ will have at least the same answers as if the query was issued against P. If two perspectives semantically subsume each other, then they are said to be equivalent.

An important aspect of perspectives is that arbitrary ontology extensions do not automatically change the semantics of existing data sources. Instead, the semantics of a data source are determined by the intended perspective, which depends on the ontology that the source commits to. Any axioms added by new ontologies are invisible from the intended perspective, and will have no effect on queries issued against it. However, an extending ontology can reuse data sources that commit to an included ontology by providing an alternate perspective on them.

3. Revisioning

While previous ontology work has usually assumed static environments or that the ontologies exist in isolation and changes will not have side effects, when ontologies are applied to the Web these assumptions must be dropped. It is inevitable that web ontologies will change, whether the purpose is to correct errors, accommodate new information, or adjust the representation of a particular domain. However, changing any of these ontologies can have far-reaching side effects because numerous web pages and ontologies may depend on them. Any attempt to coordinate ontology revisions with corresponding revisions to the dependent objects will be very expensive and likely to fail. Furthermore, in distributed environments like the Web, the dependent objects are likely to be owned by various parties who may be opposed to or ill-prepared for such revisions.

In this section we define an ontology revision as 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 the impact they would have on perspectives of existing data sources. To be succinct, we will only discuss revisions that add or remove components; the modification of a component can be thought of as a removal followed by an addition. In the rest of this section, O will refer to the original ontology, O′ to its revision, P and P′ to the perspectives formed by these respective ontologies and an arbitrary source $S=<O,D_S>$, and T and T′ to the respective theories for these perspectives.

If a revision O′ adds an arbitrary rule to ontology O, then for any source S, the perspective P′ semantically subsumes P. This proof is trivial: since the revision only adds a sentence to the corresponding theory $T' \supset T$, and by the monotonicity of first-order logic any logical consequence of T is also a logical consequence of T′. Similar reasoning is used to ascertain that if the revision removes rules, then P semantically subsumes P′. Thus, a revision that adds or removes rules can provide an alternate perspective of a legacy data source, but it may restrict or loosen the semantics that were originally intended by the author of the data.

If O′ consists of the removal of categories or relations from O, then P semantically subsumes P′. This is because there may be some atoms in S that were well-formed with respect to O that are not well-formed with respect to O′. Informally, if categories or relations are removed, predicate symbols are removed from the vocabulary. If the ground atoms of S depended on these symbols for well-formedness, then the sentences are no longer well-formed when the symbols are removed. Thus, $T' \subseteq T$ and due to the monotonicity of first order logic every logical consequence of T′ is a logical consequence of T. Revisions of this type may mean that using the revised ontology to form a perspective may result in fewer answers to a given query.

Finally, if the revision only adds categories or relations, the corresponding perspective P′ is equivalent to P. Since $T \subset T'$ it is easy to show that P′ semantically subsumes P. The proof of the other 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. It also relies on the fact that,

[2] More precisely, we mean the set of ground atoms that are logical consequences of the corresponding theory T. This is also equivalent to the least Herbrand model of T.

due to the definitions of categories and relations, the predicate of each antecedent 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 preexisting ontology. There can be no such atoms in S, since S must be well-formed with respect to some ontology $\neq O'$. Since the antecedents cannot be fulfilled, the rules will have no new logical consequences that are ground atoms. Since P semantically subsumes P′ and vice versa, P and P′ are equivalent. This result indicates that we can safely add relations or categories to the revision, and maintain the same perspective on all legacy data sources.

3.2 Versioning

As described in the previous section, different types of revisions may have different effects on the perspectives that we have on our data sources. It is these situations that make SHOE's versioning scheme important. SHOE maintains each version of the ontology as a separate web page and an instance must state which version it commits to. As a result, 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 can be used to form perspectives that semantically subsume the original perspective. 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 to form an alternate perspective of any data source. 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 will result in equivalent perspectives for 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. Ontology Divergence

As discussed earlier, an important aspect of SHOE is that interoperability is achieved through ontology reuse. That is, the preferred method of ontology development 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. As a result there will be a tendency for the most specific ontologies to diverge and become less interoperable. In these situations, occasional manual integration of ontologies is needed.

Ontology integration 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

The process for aligning ontologies can be performed either manually or semi-automatically (using tools and algorithms currently under development by other researchers). However, it is important to note that simply creating a new integrated ontology does not solve the problem of integrating information on the Web. When the web community has synthesized the ontologies (that is, other web pages and ontologies come to depend on them), all of the dependent objects would have to be revised to reflect the new ontology. Since this would be an impossible task, we instead suggest three ways to incorporate the results of an ontology integration effort, each of which is shown in Figure 2.

In the first approach, we create a mapping ontology O_M that extends the two ontologies O_1 and O_2 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. First, 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. Second, scope differences require mapping a category to the most specific category in the other domain that subsumes it. For example:

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

Third, some encoding differences can be handled by mapping individual values as in:

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

Of course, not all encodings can be mapped in SHOE, for example arithmetic functions would be needed to map meters to feet. Finally, SHOE's prefix naming mechanism obviates the need to resolve context differences. The advantage of a mapping ontology is that the domain ontologies are unchanged; thus, it can be created without the approval of the owners of the original ontology. The disadvantages are that a user must explicitly request that the mapping ontology be used to provide a new perspective and 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 used as alternate perspectives for data sources based on the domain ontologies, then data integration is automatic.

The disadvantage of the two mapping approaches is that they ignore a fundamental problem: the overlapping concepts do not belong in either domain, but 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 O_N to serve as the intersection ontology and give it a unique identifier.

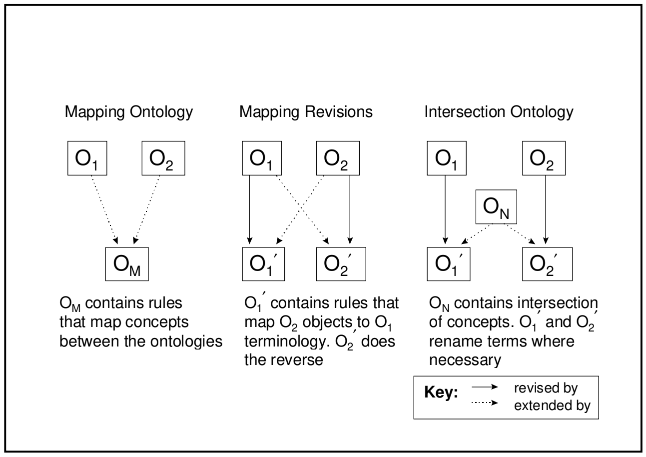

Figure 2. Integration Methods

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 O_1' and O_2' 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, we must translate this terminology to that of each domain if we wish to make existing data sources well-formed with respect to the new ontologies. 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

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. 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 for describing web resources. RDF has only a weak notion of ontologies, and although the RDF Schema proposal (Brickley and Guha 1999) improves this situation somewhat, it does not sufficiently handle the notions of revising or integrating ontologies.

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). An attempt to formalize the notion of ontology is presented in Guarino and Giaretta (1995). There is little existing ontology research that considers either distributed environments or ontology revision. The Ontolingua Server (Farquhar, Fikes, and Rice 1997) can be used for collaborative ontology development; however, it primarily deals with the construction of ontologies and not the maintenance of ontologies that have been synthesized in a distributed environment. Foo (1995) has published some initial thoughts on ontology revision.

6. Conclusions

Although other projects are using ontologies to provide some structure to Web, none of these have focused on the problem of maintaining consistency as the ontologies evolve. The main contribution of this paper is an analysis of 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 support revisions. We have developed a logic that separates data from ontologies and allows ontologies to provide different perspectives on the data. By mapping SHOE to this logic, we have shown how different types of revisions to an ontology affect the way we reason with 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 eliminate certain answers. 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, ontologies will tend to diverge and ontology integration will need to be performed periodically. We described three methods of incorporating this integration, showed how they could be applied in SHOE, and discussed the tradeoffs of each.

Although the Web is currently an untamed wilderness, 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. This work is an initial step in this direction. Future work will investigate these issues more deeply and extend the analysis to richer ontology languages.

Acknowledgments

This work was supported by the Army Research Laboratory under contract number DAAL01-97-K0135. 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.V. 1999. *RDF Schema Specification (draft)*. W3C (World Wide Web Consortium). At http://www.w3.org/TR/1999/PR-rdf-schema-19990303.html.

Farquhar, A.; Fikes, R.; and Rice, J. 1997. 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 Press.

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.

Foo, N. 1995. Ontology Revision. In *Conceptual Structures; Third International Conference*, 16-31. Berlin: Springer-Verlag.

Guarino, N. and Giaretta, P. 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.) *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, 25-32. Amsterdam: IOS Press.

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

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

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

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

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

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 *Proceedings of 1994 Monterey Workshop on Formal Methods*, 56-62. U.S. Naval Postgraduate School.