SHOE: A Prototype Language for the Semantic Web

Jeff Heflin
James Hendler
Sean Luke

Department of Computer Science
University of Maryland,
College Park, MD 20742, USA

July 16, 2001

Abstract

The term *Semantic Web* was coined by Tim Berners-Lee to describe his proposal for "a web of meaning," as opposed to the "web of links" that currently exists on the Internet. To achieve this vision, we need to develop languages and tools that enable machine understandable web pages. The SHOE project, begun in 1995, was one of the first to begin exploring these issues. In this paper, we describe our experiences developing and using the SHOE language. We begin by describing the unique features of the World Wide Web and how they must influence potential Semantic Web languages. We then discuss why web standards such as XML and RDF are currently insufficient for the Semantic Web. We present SHOE, a language which allows web pages to be annotated with semantics, describe its syntax and semantics, and discuss our approaches to handling characteristics of the Web such as distributed authority and rapid evolution. We discuss the implementation issues of such a language, and describe some generic tools that we have built to aid in its use. Finally, we demonstrate the language and tools by applying them to two different domains. The language, tools, and details of the applications are all available on the World Wide Web at *http://www.cs.umd.edu/projects/plus/SHOE/*.

1 Introduction

The World Wide Web is a vast repository of information, but its utility is restricted by limited facilities for searching and integrating this information. The problem of making sense of the Web has engaged the minds of numerous researchers from fields such as databases, artificial intelligence, and library science; and these researchers have applied numerous approaches in an attempt to solve it. Tim Berners-Lee, inventor of the Web, has coined the term *Semantic Web* to describe a vision of the future in which the "web of links" is replaced with a "web of meaning." In this paper, we examine the thesis that the "the Semantic Web can be achieved if we describe web resources in a language that makes their meaning explicit."

Although a Semantic Web language should draw on the research conducted in knowledge representation, it should also take into account the very nature of the Web. Let's consider some of the issues that arise:

- **The Web is distributed.** One of the driving factors in the proliferation of the Web is the freedom from a centralized authority. However, since the Web is the product of many individuals, the lack of central control presents many challenges for reasoning with its information. First, different communities will use different vocabularies, resulting in problems of synonymy (when two different words have the same meaning) and polysemy (when the same word is used with different meanings). Second, an intelligent web agent simply cannot assume that all of the information has been entered solely under a knowledge engineer's watchful eye and is therefore correct and consistent. The lack of editorial review or quality control means that each page's reliability must be questioned: there are quite a number of well-known "web hoaxes" where information was published on the Web with the intent

to amuse or mislead. Furthermore, since there can be no global enforcement of integrity constraints on the Web, information from different sources may be in conflict. Some of these conflicts may be due to philosophical disagreement; different political groups, religious groups, or nationalities may have fundamental differences in opinion that will never be resolved. Any attempt to prevent such inconsistencies must favor one opinion, but the correctness of the opinion is very much in the "eye of the beholder."

- **The Web is dynamic.** The web changes at an incredible pace, much faster than a user or even a "softbot" web agent can keep up with. While new pages are being added, the content of existing pages is changing. Some pages are fairly static, others change on a regular basis and still others change at unpredictable intervals. These changes may vary in significance: although the addition of punctuation, correction of spelling errors or reordering of a paragraph does not affect the semantic content of a document, other changes may completely alter meaning, or even remove large amounts of information. A web agent must assume that its data can be, and often will be, out of date.

  The rapid pace of information change on the Internet poses an additional challenge to taxonomy and ontology designers. Without a reasonably unifying ontological framework, knowledge on the web balkanizes, and web agents will struggle to learn and internally cross-map a myriad of incompatible knowledge structures. But an imposed unifying framework risks being too inflexible to accommodate new topics, new ideas, and new knowledge rapidly entering the Web.

- **The Web is massive.** Recent estimates place the number of indexable web pages at over 2 billion and this number is expected to double within a year. Even if each page contained only a *single* piece of agent-gatherable knowledge, the cumulative database would be large enough to bring most reasoning systems to their knees. To scale to the size of the ever growing Web, we must either restrict expressivity of the language or use incomplete reasoning algorithms.

- **The Web is an open world.** A web agent is not free to assume it has gathered all available knowledge; in fact, in most cases an agent should assume it has gathered rather little available knowledge. Even the largest search engines have only crawled about 25% of the available pages. However, in order to deduce more facts, many reasoning systems make the closed-world assumption. That is, they assume that anything not entailed in the knowledge base is not true. Yet it is clear that the size and evolving nature of the Web makes it unlikely that any knowledge base attempting to describe it could ever be complete.

In an attempt to deal with these issues, we have designed a language named **SHOE**, for Simple HTML Ontology Extensions.[1] SHOE is one of the first languages that allows ontologies to be designed and used directly on the World Wide Web [31]. The purpose of this paper is to identify the critical issues in designing the Semantic Web and and to describe initial steps towards solutions. We describe work that influenced SHOE, present an overview of the language, describe its syntax and semantics, and discuss how SHOE addresses the issues posed in this introduction. We then discuss the problem of implementing a system that uses SHOE, focusing on the query engine aspect of the system. Then we describe some generic tools (applets and class libraries) that make SHOE more usable in the web environment. We describe some current applications of SHOE designed to show its applicability, and then discuss some lessons learned from these implementations, concluding with some directions for future work.

2 Background

There are severals areas that can serve as a foundation for a Semantic Web language. Markup languages such as HTML and XML have great support in the Web community, and their role in the Semantic Web must be considered. The field of knowledge representation is directly concerned with the issue of semantics, and

[1] There are several reasons we chose this name. In the spirit of early KR languages, we wanted an acronym that was also a natural language term. In a spirit of "putting our money where our mouth is," we wanted a word which could not be searched for on the web without some sort of ontological context – at the time this paper is being written, Google finds 1,900,000 pages containing the word "shoe." And in the spirit of putting some of the fun back into AI, we wanted to refer to the web agents we define using this language as really "kicking butt." Thus, this acronym was an obvious choice.

has resulted in many languages from which ideas can be drawn. Work in deductive databases has studied reasoning with large amounts of data, and the datalog language can be used for inspiration. Finally, research into ontology can offer insights into reusable, modularized knowledge components. This section discusses each of these areas in more detail. It should be noted that although approaches such as information retrieval, wrappers, semi-structured databases, machine learning, and natural language processing have been applied to the problem of querying and understanding the Web, they do not directly relate to the design of a language for the Semantic Web and thus will not be covered here.

2.1 SGML

The Standard Generalized Markup Language (SGML) [24] is a language that allows special codes to be embedded in a text data stream. These codes, also called tags, can provide additional information about the text, such as indicating that a certain word should be emphasized. The term *element* is used to describe a start-tag, a end-tag and the data contained between them. An *attribute* can be included in a start-tag to include additional information about the element. SGML is "generalized" because it allows one to define the elements and attributes that describe a specific markup language. This information is contained in a document type definition (DTD) that specifies valid elements, the contents of these elements, and which attributes may modify an element. The benefits of SGML include platform independence, separation of content from format, and the ability to determine if documents conform to structural rules.

2.2 HTML

HTML is commonly thought of as the language of the Web. It introduced many people to the syntax of SGML, but can be thought of as an SGML application: a markup language that can be described using SGML. Originally, HTML was concerned mostly with presentation. Besides the all important anchor tag that gives HTML its hypertext character, it includes tags to indicate paragraphs, headings, lists, etc. HTML 2.0 [3] introduced a number of weak semantic markup mechanisms. The META element specifies meta-data in the form of a name/value pair. The REL attribute of the anchor and link elements names a relationship from the enclosing document to the document pointed to by a hyperlink; the REV attribute names the relationship in the reverse direction. HTML 3.0 [36] added the CLASS attribute, which could be used within almost any tag to create semantic subclasses of that element, Unfortunately, the semantic markup elements of HTML are rarely used, and even if they were more widely accepted could only establish relationships along hypertext links (using <LINK> or <A>).

To address the semantic limitations of HTML, Dobson and Burrill [15] attempted to reconcile it with the Entity-Relationship (ER) database model. This is done by supplementing HTML with a simple set of tags that define "entities" within documents, labeling sections of the body text as "attributes" of these entities, and defining relationships from an entity to outside entities.

2.3 XML

The World Wide Web Consortium (W3C) developed the Extensible Markup Language (XML) [8] to serve as a simplified version of SGML for the Web. Like SGML, XML can use DTDs to ensure that documents conform to a common grammar.[2] Thus a DTD provides a syntax for an XML document, but the semantics of a DTD are implicit. That is, the meaning of an element in a DTD is either inferred by a human due to the name assigned to it, is described in a natural-language comment within the DTD, or is described in a document separate from the DTD. Humans can then build these semantics into tools that are used to interpret or translate the XML documents, but software tools cannot acquire these semantics independently. Thus, an exchange of XML documents works well if the parties involved have agreed to a DTD beforehand, but becomes problematic when one wants to search across the entire set of DTDs or to spontaneously integrate information from multiple sources.

One of the hardest problems in any integration effort is mapping between different representations of the same concepts – the problem of integrating DTDs is no different. One difficulty is identifying and mapping

[2] A document that does so is said to be valid, while documents which are consistent with XML's syntax but do not have a DTD are said to be well-formed.

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

<!-- The NAME is a subelement with element content -->
<PERSON>
   <NAME><FNAME>John</FNAME><LNAME>Smith</LNAME></NAME>
</PERSON>

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

Figure 1: Structural Differences in Representation

differences in naming conventions. As with natural language, XML DTDs have the problems of polysemy and synonymy. For example, the elements <PERSON> and <INDIVIDUAL> might be synonymous. Similarly, an element such as <SPIDER> might be polysemous: in one document it could mean a piece of software that crawls the World Wide Web while in another it means an arachnid that crawls a web of the silky kind. Furthermore, naming problems can apply to attribute names just as easily as they apply to element names. In general, machines do not have access to the contextual information that humans have, and thus even an automated dictionary or thesaurus would be of little help in resolving the problems with names described here.

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

Once humans have identified the appropriate mappings between two DTDs, it is possible to write XSL Transformations (XSLT) stylesheets [12] that can be used to automatically translate one document into the format of another. Although this is a good solution to the integration problem when only a few DTDs are relevant, it is unsatisfactory when there are many DTDs; if there are n DTDs, then there would need to be $O(n^2)$ different stylesheets to allow automatic transformation between any pair of them. Furthermore, when a DTD was created or revised, someone would have to create or revise the n stylesheets to transform it to all other DTDs. Obviously, this is not a feasible solution.

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

2.4 Knowledge Representation

One of the venerable sub-fields of artificial intelligence is that of knowledge representation (KR). The goal of KR is to provide structures that allow information to be stored, modified, and reasoned with, all in an efficient manner. Thus, a good KR language is expressive, concise, unambiguous and independent of context, while systems built upon the language should be able to acquire information and perform useful inferences efficiently. From the very beginnings of AI, KR has been crucial to the pursuit, and the field has remained an active and important research area spawning entire sub-disciplines of its own. Early languages, such as

KL-ONE [6] and KRL [4] have evolved into modern powerhouses like LOOM [32] , Classic [7], and CYC-L [29].

One of the oldest knowledge representation formalisms is semantic networks. A semantic net represents knowledge as a set of nodes connected by directed links; essentially it can be described by a directed acyclic graph. In such a representation, meaning is implied by the way a concept is connected to other concepts. Frame systems are another representation that is isomorphic to semantic networks. In the terminology of such systems, a frame is a named data object that has a set of slots, where each slot represents a property or attribute of the object. Slots can have one or more values; these values may be pointers to other frames. KRL [4] is an example of an early KR language based on frame systems.

Advanced semantic networks and frame systems typically include the notion of abstraction, which is represented using *is-a* and *instance-of* links. An *is-a* link indicates that one class is included within another, while an *instance-of* link indicates that an individual is a member of a class. These links have correlations in basic set theory: *is-a* is like the subset relation and *instance-of* is like the element of relation. The collection of *is-a* links specifies a partial order on classes; this order is often called a taxonomy or categorization hierarchy. The taxonomy can be used to generalize a concept to a more abstract class or to specialize a class to its more specific concepts. As demonstrated by the popularity of Yahoo and the Open Directory, taxonomies are clearly useful for aiding a user in locating relevant information on the Web.

A more recent KR formalism is description logic, which grew out of the work on KL-ONE [6]. In a description logic, definitions of terms are formed by combining concepts and roles that can provide either necessary and sufficient conditions or just necessary conditions. A description is said to subsume another if it describes all of the instances that are described by the second description. An important feature of description logic systems is the ability to perform automatic classification, that is, automatically insert a given concept at the appropriate place in the taxonomy. The advantages of descriptions logics are they have well-founded semantics and the factors that affect their computational complexity are well understood, but it is unclear whether their inferential capabilities are the right ones for the Web.

In general, knowledge representation can offer the Semantic Web insight into the design of semantic languages and the development of reasoning methods for them. However, most KR systems do not scale well, and would thus be unable to support data gathered from even a tiny portion of the Web. Additionally, KR research often assumes a centralized knowledge base with a single controlling authority; obviously this assumption must be dropped when dealing with the Web.

2.5 RDF

The Resource Description Framework (RDF) [28] is a recommendation endorsed by the W3C that attempts to address XML's semantic limitations. Technically, RDF is not a language, but a data model of metadata instances. To include RDF in files, its designers have chosen a frame-like, XML syntax although they emphasize this is only one of many possible representations of the RDF model. RDF has a number of abbreviated syntactical variations which is an advantage for content providers but requires more machinery in RDF parsers. Since these syntaxes all have a well-defined mapping into the data model, they avoid some of the problems with representational choices in basic XML. Nevertheless, it is still easy to create different representations for a concept.

The RDF data model is little more than a semantic network without inheritance; it consists of nodes connected by labeled arcs, where the nodes represent web resources and the arcs represent attributes of these resources. Since RDF is based on semantic networks, it is inherently binary. Of course, any n-ary relation can be expressed as a set of binary relations. Furthermore, authors can use an abbreviated form of the RDF syntax to express the values for many properties at once, essentially simulating an n-ary relation.

To allow for the creation of controlled, sharable, extensible vocabularies the RDF working group has developed the RDF Schema Specification [10]. This specification defines a number of properties that have specific semantics. The property rdf:type is used to express that a resource is a member of a given class, while the property rdfs:subClassOf essentially states that one class is a subset of another. These properties are equivalent to the *instance-of* and *is-a* links that have been used in AI systems for decades. With the rdfs:subClassOf property, schema designers can build taxonomies of classes for organizing their resources. RDF Schema also provides properties for describing properties; the property rdfs:subPropertyOf allow properties to be specialized in a way similar to classes, while the properties rdfs:domain and rdfs:range allow constraints to

be placed on the domain and range of a property.

Since it is possible that different schemas may use the same strings to represent different conceptual meanings, RDF uses XML namespaces [9] to assign a separate namespace to each schema. This approach has two disadvantages. First since namespaces can be used with any element and RDF schemas need not be formally specified, it is possible to write RDF statements such that it is ambiguous as to whether certain tags are RDF or intermeshed tags from another namespace. Second, each set of RDF statements must explicitly specify the namespace for every schema that is referenced by one of the statements, even for schemas that were extended by a schema whose namespace has already been specified.

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

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

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

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

2.6 Datalog

Datalog is a language frequently used in describing deductive databases. It is similar to Prolog in that it consists entirely of Horn clauses, but differs in that it does not allow function symbols and is a strictly

declarative language.[3] Datalog is based on the relational model but defines two types of relations: *extensional database* (EDB) relations are those predicates which are physically stored in the database, while *intensional database* (IDB) relations are those that can be computed from a set of logical rules.

Datalog restricts its horn clauses to be *safe*, meaning that all of its variables are *limited*. Datalog defines "limited" as follows: variables are limited if they appear in an ordinary predicate of the rule's body, appear in an '=' comparison with a constant, or appear in an '=' comparison with another limited variable. Datalog's Horn clauses may depend on each other recursively. Datalog allows negation in a limited form called *stratified negation*, which we will not discuss here.

Datalog is relevant to the design of a Semantic Web language because it allows important classes of rules to be expressed while recent optimizations such as *magic sets*, which combine forward and backward chaining, have made reasoning more efficient. Additionally, it seems reasonable to expect that the Web will consist of a large EDB and a comparatively small IDB, which is an ideal situation for a datalog system.

2.7 Ontology

The term ontology, which is borrowed from philosophy, is defined as "a particular theory about being or reality." As such, an ontology provides a particular perspective onto the world, or some part there of. Where a knowledge representation system specifies how to represent concepts, an ontology specifies what concepts to represent and how they are interrelated. Most researchers agree that an ontology must include a vocabulary and corresponding definitions, but it is difficult to achieve consensus on a more detailed characterization. Typically, the vocabulary includes terms for classes and relations, while the definitions of these terms may be informal text, or may be specified using a formal language like predicate logic. The advantage of formal definitions is that they allow a machine to perform much deeper reasoning; the disadvantage is that these definitions are much more difficult to construct.

Numerous ontologies have been constructed, with varying scopes, levels of detail, and viewpoints. Noy and Hafner [34] provide a good overview and comparison of some of these projects. One of the more prominent themes in ontology research is the construction of reusable components. The advantages of such components are clear: large ontologies can be quickly constructed by assembling and refining existing components, and integration of ontologies is easier when the ontologies share components. One of the most common ways to achieve reusability is to allow the specification of an inclusion relation that states that one or more ontologies are included in the new theory. If these relationships are acyclic and treat all elements of the included ontology as if they were defined locally then an ontology can be said to extend its included ontologies. This is the case for most systems, however Ontolingua [18] has the most powerful features for reusability: inclusion relations that may contain cycles, the ability to restrict axioms, and polymorphic refinement.

The largest ontology effort is Cyc [29], an ongoing project with the ambitious goal of encoding the entirety of common sense. An essential component of Cyc is the partitioning of its knowledge into microtheories, which can extend one another using standard ontology inclusion principles. Since Cyc has an enormous ontology, microtheories are essential to its creation. They simplify the encoding of assertions by knowledge engineers, avoid the inevitable contradictions that arise from a large knowledge base, and help guide inferencing mechanisms by grouping relevant statements.

We contend that ontologies can be used on the Web to help structure the information – but only if we design the language to take into account the characteristics of the Web described in the Introduction. All information is presented in some context. When people read documents, they draw on their knowledge of the domain and general language to interpret individual statements. Context is often required to disambiguate terms and to provide a background framework for understanding. Ontologies provide a mechanism by which context information can be specifically encoded, and a Semantic Web language must allow this information to be specified on web pages or in other repositories that refer to web-based information. Additionally, the context can be used to resolve the problem of polysemy, while mappings between ontologies can be used to resolve synonymy. Extending XML-like languages to include ontology features will allow far more structuring, and by adding inferential capabilities we allow for knowledge collected from distributed sources to be "fused."

[3] Prolog is not strictly declarative because the order of the rules determines how the system processes them.

3 The SHOE Language

SHOE combines features of markup languages, knowledge representation, Datalog, and ontologies in an attempt to address the unique problems of semantics on the Web. It supports knowledge acquisition on the Web by supplementing HTML's presentation oriented tags with tags that provide semantic meaning. The basic structure consists of *ontologies*, which are entities that define rules guiding what kinds of assertions may be made and what kinds of inferences may be drawn on ground assertions, and *instances*, which are entities that make assertions based on those rules. As a knowledge representation, SHOE borrows characteristics from both predicate logics and frame systems.

The original syntax of SHOE was greatly influenced by HTML, which made it natural to include SHOE in HTML documents and eased the learning curve for web authors who wished to use SHOE. We later defined an SGML DTD for SHOE that is derived from the formal HTML DTD, thus making SHOE a formal application of SGML while maintaining an HTML-compatible syntax. Due to the similarities between SGML and XML, it was easy to create a slight variant of the SHOE syntax that is compatible with XML. However, since most websites are still written in HTML (which is not compatible with XML), the SGML version of SHOE remains the standard for implementation, but websites that have begun to migrate to XML (by using XHTML or another XML application) can use the SHOE XML DTD. When XML becomes ubiquitous, the standard version of SHOE will be the XML variant.

There are a number of advantages to using an XML syntax for SHOE. First, although more standard KR syntaxes, such as first-order logic or S-expressions, could be embedded between a pair of delimiting tags, these would be even more foreign to the average web user than SHOE's syntax, which at least has a familiar format. Second, the XML syntax allows SHOE information to be analyzed and processed using the Document Object Model (DOM), allowing software that is XML-aware, but not SHOE-aware to still use the information in more limited but nevertheless powerful ways. For example, some web browsers are able to graphically display the DOM of a document as a tree, and future browsers will allow users to issue queries that will match structures contained within the tree. The third reason for using an XML syntax is that SHOE documents can then use the XSLT stylesheet standard [12] to render SHOE information for human consumption. This is perhaps the most important reason for an XML syntax, because it can eliminate the redundancy of having a separate set of tags for the human-readable and machine-readable knowledge.

In this section, we provide a brief description of the syntax of the language followed by a formal model and a discussion of SHOE's key features. The complete syntax is presented as an XML DTD in Appendix A. The interested reader can find a detailed description of SHOE's syntax in the SHOE Specification [30].

3.1 SHOE Ontologies

SHOE uses ontologies to define the valid elements that may be used in describing entities. Each ontology can reuse other ontologies by extending them. An ontology is stored in an HTML file and is made available to document authors and SHOE agents by placing it on a web server. This file includes tags that identify the ontology, state which ontologies (if any) are extended, and define the various elements of the ontology. Figure 2 shows an example of a SHOE ontology.

In SHOE syntax, an ontology appears between the tags <ONTOLOGY ID=*id* VERSION=*version*> and </ONTOLOGY> which is identified by the combination of the *id* and *version*. An ontology can define categories, relations and other components by including special tags for these purposes.

The tag <DEF-CATEGORY>) can be used to make *category definitions* that specify the categories (also called classes) under which various instances could be classified. Categories may be grouped as subcategories under one or more supercategories, essentially specifying the *is-a* relation that is commonly used in semantic networks and frame systems. The use of categories allows taxonomies to be built from the top down by subdividing known classes into smaller sets. The example ontology defines many categories, including GraduateStudent, which is a subcategory of both Student and Worker.

The tag <DEF-RELATION> (which is closed by a </DEF-RELATION> tag) can be used to make *relational definitions* that specify the kinds of relational assertion that may be made by instances regarding instances and other data. Each relation has some fixed number of arguments, and the type of each argument is either a category or one of four basic data types (string, number, date, or boolean value). One of the relationships defined by the example ontology is advises, which is between an instance of category GraduateStudent and an

```
<HTML>
<HEAD>
    <TITLE>University Ontology</TITLE>
Tell agents that we're using SHOE
    <META HTTP-EQUIV="SHOE" CONTENT="VERSION=1.0">
</HEAD>
<BODY>
Declare an ontology called "university-ontology".
    <ONTOLOGY ID="university-ontology" VERSION="1.0">
Borrow some elements from an existing ontology, prefixed with a "b."
        <USE-ONTOLOGY ID="base-ontology" VERSION="1.0" PREFIX="b"
            URL="http://www.cs.umd.edu/projects/plus/SHOE/base.html">
Define some categories and subcategory relationships
        <DEF-CATEGORY NAME="Person" ISA="b.SHOEentity">
        <DEF-CATEGORY NAME="Organization" ISA="b.SHOEentity">
        <DEF-CATEGORY NAME="Worker" ISA="Person">
        <DEF-CATEGORY NAME="Advisor" ISA="Worker">
        <DEF-CATEGORY NAME="Student" ISA="Person">
        <DEF-CATEGORY NAME="GraduateStudent" ISA="Student Worker">
Define some relations; these examples are binary, but relations can be n-ary
        <DEF-RELATION NAME="advises">
            <DEF-ARG POS="1" TYPE="Advisor">
            <DEF-ARG POS="2" TYPE="GraduateStudent"></DEF-RELATION>
        <DEF-RELATION "gpa">
            <DEF-ARG POS="1" TYPE="Student">
            <DEF-ARG POS="2" TYPE="b.NUMBER"></DEF-RELATION>
        <DEF-RELATION "suborganization">
            <DEF-ARG POS="1" TYPE="Organization">
            <DEF-ARG POS="2" TYPE="Organization"></DEF-RELATION>
        <DEF-RELATION "works-for">
            <DEF-ARG POS="1" TYPE="Person">
            <DEF-ARG POS="2" TYPE="Organization"></DEF-RELATION>
Define a transfers-through inference over working for organizations
        <DEF-INFERENCE>
            <INF-IF>
                <RELATION NAME="works-for">
                    <ARG POS="1" VALUE="x" VAR>
                    <ARG POS="2" VALUE="y" VAR></RELATION>
                <RELATION NAME="suborganization">
                    <ARG POS="1" VALUE="y" VAR>
                    <ARG POS="1" VALUE="z" VAR></RELATION></INF-IF>
            <INF-THEN>
                <RELATION NAME="works-for">
                    <ARG POS="1" VALUE="x" VAR>
                    <ARG POS="2" VALUE="z" VAR></RELATION></INF-THEN>
        </DEF-INFERENCE>
    </ONTOLOGY>
</BODY>
</HTML>
```

Figure 2: An Ontology Example

instance of category Advisor. Similarly, the relationship gpa defines a relation that exists between a Student and a number.

SHOE uses inference rules, indicated by the <DEF-INFERENCE> tag, to supply additional axioms. A SHOE inference rule consists of a set of antecedents (one or more subclauses describing assertions that entities might make) and a set of consequents (consisting of one or more subclauses describing an assertion that may be inferred if all assertions in the body are made). The <INF-IF> and <INF-THEN> tags indicate the antecedents and consequents of the inference, respectively. There are three types of inference subclauses: category, relation and comparison. A category clause is satisfied if the instance is member of a specified category, a relation subclause is satisfied if the relation is entailed by existing assertions, and a comparison subclause is satisfied if a particular mathematical relation (such as less-than or equal-to) holds between a pair of values. Although category and relation subclauses can appear in both the antecedents and consequents, comparison clauses can only appear in the antecedents. 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. All variables must be limited, and the type of each variable, which can be determined from its use it category or relation clauses, must be consistent. The ontology in the example specifies that working for organizations transfers through to superorganizations, that is, $(\forall x \in \text{Worker})\ (\forall y \in \text{Organization})\ (\forall z \in \text{Organization})\ \text{works-for}(x,y) \wedge \text{suborganization}(y,z) \Rightarrow \text{works-for}(x,z)$.

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

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

3.2 SHOE Instances

Unlike RDF, SHOE makes a distinction between what can be said in an ontology and what can be said on an arbitrary web page. Ordinary web pages declare one or more instances that represent SHOE entities, and each instance describes itself or other instances using categories and relations. An example of a SHOE instance is shown in Figure 3. The syntax for instances includes an <INSTANCE> element that has an attribute for a KEY that uniquely identifies the instance. We recommend that the URL of the web page be used as this key, since it is guaranteed to identify only a single resource. An instance commits to a particular ontology by means of the <USE-ONTOLOGY> tag, which has the same function as the identically named element used within ontologies. To prevent ambiguity in the declarations, ontology components are referred to using the prefixing mechanism described earlier. The use of common ontologies makes it possible to issue a single logical query to a set of data sources and enables the integration of related domains. Additionally, by specifying ontologies the content author indicates exactly what meaning he associates with his assertions, and does not need to worry that an arbitrary definition made in some other ontology will alter this meaning.

An instance contains ground *category assertions* and *relation assertions* made by it. A category assertion is made within the <CATEGORY NAME=*y* FOR=*x*> tag, and says that the instance claims that some instance x should be categorized under category y. In the example, the instance http://univ.edu/john claims that http://univ.edu/mary is an Advisor.

A relational assertion is enclosed by the <RELATION NAME=*foo*> and </RELATION> tags, and says that the instance claims that an n-ary relation *foo* exists between some n number of appropriately typed arguments consisting of data or instances. In the example, the instance http://univ.edu/john claims that there exists the relation advises between http://univ.edu/john and his advisor http://univ.edu/mary and that that the

```
<HTML>
<HEAD>
    <TITLE>John's Web Page</TITLE>
Tell agents that we're using SHOE
    <META HTTP-EQUIV="SHOE" CONTENT="VERSION=1.0">
</HEAD>
<BODY>
    <P>This is my home page, and I've got some SHOE data on it about me and my advisor. Hi, Mom!</P>
Create an Instance. There's only one instance on this web page, so we might as well use the web page's URL as its key.
If there were more than one instance, perhaps the instances might have keys of the form http://univ.edu/john#FOO
    <INSTANCE KEY="http://univ.edu/john">
Use the semantics from the ontology "university-ontology", prefixed with a "u."
    <USE-ONTOLOGY ID="university-ontology" VERSION="1.0" PREFIX="u"
        URL="http://univ.edu/ontology">
Claim some categories for me and others.
    <CATEGORY NAME="u.GraduateStudent">
    <CATEGORY NAME="u.Advisor" FOR="http://univ.edu/mary">
Claim some relationships about me and others. "me" is a keyword for the enclosing instance.
    <RELATION NAME="u.advises">
        <ARG POS="1" VALUE="http://univ.edu/mary">
        <ARG POS="2" VALUE="me"> </RELATION>
    <RELATION NAME="u.gpa">
        <ARG POS="1" VALUE="me">
        <ARG POS="2" VALUE="3.8"> </RELATION>
    </INSTANCE>
</BODY>
</HTML>
```

Figure 3: An Instance Example

gpa of http://univ.edu/john is 3.8.

3.3 SHOE's Semantics

In order to describe the semantics of SHOE, we will extend a standard model-theoretic approach for definite logic with mechanisms to handle distributed ontologies. For simplicity, this model intentionally omits those features of the SHOE language that are rarely used.

We define an ontology O to be a tuple $< V, A >$ where V is the vocabulary and A is the set of axioms that govern the theory. Formally, V is a set of predicate symbols, each with some arity ≥ 0 and distinct from symbols in other ontologies,[4] while A is a set of definite program clauses that have the standard logical semantics.[5] We now discuss the contents of 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. Due to the assumption that names must be unique, name conflicts can be ignored.

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-RELATION> statement adds a symbol to the vocabulary and, depending on the data types of arguments, possibly some number of axioms. If an argument has a data type is a basic data type (such as number), then it is simply used to check the syntax of the data and to perform comparisons. In this case, no additional axioms are added. However, if the argument has a data type that is a category, then there is 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 ith argument is C, then $[R(x_1, ..., x_i, ...x_n) \rightarrow C(x_i)] \in A$. This rule is a consequence of SHOE's open-world policy: since there is no way to know that a given object in a relation assertion is *not* a member of a category appropriate for that relation, it is better to assume that this information is yet undiscovered than it is to assume that the relation is in error. Note that this is in contrast to arguments that are basic data types, where type checking is performed to validate the relation. Basic data types are treated differently because they *are* different. They have syntax which can be checked in ways that category types cannot, which allows us to impose stringent input-time type checking on basic data types.

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. For comparison clauses, it is assumed that there exists built-in predicates that can compare each of the basic data types.

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 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.

Now we turn our attention to data sources, such as one or more web pages, that use an ontology to make relation and category assertions. 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 assertions. 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 wish to be able to describe the meaning of a given data source, but it is important to realize that on the Web, the same data could have different meanings for different people. An agent may be able to draw useful inferences from a data source without necessarily agreeing with the ontology intended by the data's author. A common case would be when an agent wishes to integrate information that depends on two overlapping but still distinct ontologies. Which set of rules should the agent use to reason about this data?

[4] In actuality, SHOE has a separate namespace for each ontology, but one can assume that the symbols are unique because it is always possible to apply a renaming that appends a unique ontology identifier to each symbol.

[5] A definite program clause is a Horn clause that has at least one antecedent and exactly one consequent.

There are many possible answers, and we propose that the agent should be free to choose. To describe this notion, we define 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 P is the *intended perspective*, otherwise it is an *alternate perspective*. If there are elements of D_S that are not well-formed with respect to O, these elements are considered to be irrelevant to the perspective. If W_S is the subset of D_S that is well-formed with respect to O, then P is said to result in a definite logic theory $T = W_S \cup A$.

Finally, we can describe the semantics of a perspective P using a model theoretic approach. 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. We define a query on P as a Horn clause with no consequent that has the semantics typically assigned to such queries for a definite logic program T.

We also introduce one additional piece of terminology that will be used later in the paper. If every ground atomic logical consequence of perspective P is also a ground atomic logical consequence of perspective P' then P' is said to *semantically subsume* P. In such cases, 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.

3.4 Discussion

SHOE was designed specifically with the needs of distributed internet agents in mind. In particular, it is designed to enable interoperability in environments with multiple authors, where the underlying representations are likely to change, and where the amount of data is voluminous. In this section, we discuss how SHOE addresses each of these issues.

3.4.1 Interoperability in Distributed Environments

SHOE attempts to maximize interoperability through the use of shared ontologies, prefixed naming, prevention of contradictions, and locality of inference rules. This section discusses each of these in turn.

Figure 4 shows how the ontology extension and renaming features of the language promote interoperability. When two ontologies need to refer to a common concept, they should both extend an ontology in which that concept is defined. In this way, consistent definitions can be assigned to each concept, while still allowing communities to customize ontologies to include definitions and rules of their own for specialized areas of knowledge. These methods allow the creation of high-level, abstract unifying ontologies extended by often-revised custom ontologies for specialized, new areas of knowledge. There is a trade-off between trust of sources far down in the tree (due to their fleeting nature) and the ease of which such sources can be modified on-the-fly to accommodate new important functions (due to their fleeting nature). In a dynamic environment, an ontology too stable will be too inflexible; but of course an ontology too flexible will be too unstable. SHOE attempts to strike a balance using simple economies of distribution.

The problems of synonymy and polysemy are handled by the extension mechanism and <DEF-RENAME> tag. Using this tag, ontologies can create aliases for terms, so that domain-specific vocabularies can be used. For example, in Figure 4, the term WebBot in internet-ont2 means the same thing as Spider in internet-ont due to a <DEF-RENAME> tag in internet-ont2. Although the extension and aliasing mechanisms solve the problem of synonymy of terms, the same terms can still be used with different meanings in different ontologies. This is not undesirable, a term should not be restricted for use in one domain simply because it was first used in a particular ontology. As shown in Figure 4, in SHOE different ontologies may also use the same term to define a different concept. Here, the term Spider means different things in internet-ont and bug-ont because the categories have different ancestors. To resolve any ambiguity that may arise, ontological elements are always referenced using special prefixes that define unique paths to their respective enclosing ontologies. Instances and ontologies that reference other ontologies must include statements identifying which ontologies are used and each ontology is assigned a prefix which is unique within that scope. All references to elements from that ontology must include this prefix, thereby uniquely identifying which definition is desired.

In the case of instances, each must be assigned a key; SHOE's protocol further allows agents on the web to guarantee key uniqueness by including in the key the URL of the instance in question. In SHOE, it is

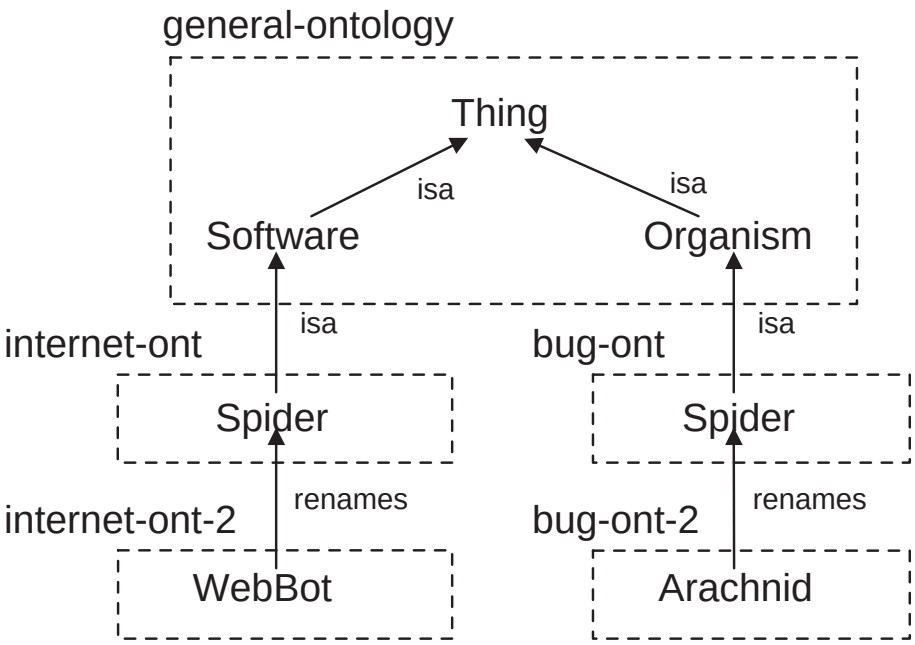

Figure 4: Ontology Interoperability

assumed that each key identifies exactly one entity, but no assumptions are made about whether two keys might identify the same entity. This is because many different URLs could be used to refer to the same page, due to the facts that a single host can have multiple domain names and operating systems may allow many different paths to the same file. To solve these problems in a practical setting, a canonical form can be chosen for the URL; an example rule might be that the full path to the file should be specified, without operating systems shortcuts such as '~ ' for a user's home directory. Even then, there are still problems with multiple keys possibly referring to the same conceptual object. At any rate, this solution ensures that the system will only interpret two objects as being equivalent when they truly are equivalent. Ensuring that two object references are matched when they conceptually refer to the same object is an open problem.

In distributed systems, a contradiction cannot be handled by simply untelling the most recent assertion, otherwise the system would give preference to those authors who provided their information first, regardless of whether it was true, false or a matter of opinion. Rather than delve into complex procedures for maintaining consistency, we chose to keep SHOE easy to understand and implement. Therefore, we have carefully designed the language to eliminate the possibility of contradictions between agent assertions. SHOE does this in four ways:

1. SHOE only permits assertions, not retractions.
2. SHOE does not permit logical negation.
3. SHOE does not have single-valued relations, that is, relational sets which may have only one value (or some fixed number of values).
4. SHOE comparisons only appear in the antecedents of inference rules, so that it is impossible to infer a fact that contradicts the inherent sorting of a data type.

Although this restricts the expressive power of the language, in our practical experience, we have not yet found it to be a significant problem. Databases have provided useful services to many organizations and

individuals despite having far less expressive power than SHOE. The intent of SHOE is to provide a solution that works efficiently in large, distributed data situations.

It should be noted that SHOE does not prevent "contradictions" that are not logically inconsistent. If source A says $father(Mark, Katherine)$ and source B says $father(Katherine, Mark)$, the apparent contradiction is because one source is misusing the $father$ relation. However, this does not change the fact that A and B made those assertions. A similar problem may occur in an ontology where an inference rule derives a conclusion whose interpretation would be inconsistent with another ontology. Therefore, it is the ontology designer's responsibility to make sure that the ontology is correct and that it is consistent with all ontologies that it extends. It is expected that ontologies which result in erroneous conclusions will be avoided by users, and will thus be weeded out by natural selection.

Yet another problem with distributed environments is the potential interference of rules created by other parties: a rule created by one individual could have unwanted side-effects for other individuals. For these reasons, SHOE only allows rules to be defined in ontologies, and the only rules that could apply to a given assertion are those which are defined in the ontologies used by the instance making the assertion. Since rules can only be expressed in ontologies, the process of determining when a rule is applicable is simplified, and page authors can use this to control the side-effects of their assertions. If a user wishes to view an instance in a different context or use it in ways not originally intended by the author, then the user can use an alternate perspective for the instance that is based on a different, but compatible ontology.

3.4.2 Ontology Evolution

The Web's changing nature means that ontologies will have to be frequently changed to keep up with current knowledge and usage. Since physically revising an ontology can invalidate objects that reference it for vocabulary and definitions, it is useful to think of a revision as a new ontology that is a copy of the original ontology to which modifications have been made. In fact, this is exactly what SHOE does: each version of an ontology is a separate file and is assigned a unique version number, while all references to an ontology must denote a specific version. How then, is a revision different from an ontology with a different identifier? The answer is that a revision can specify that it is backwardly-compatible with an earlier version (using the backward-compatible-with attribute of the ontology), which allows interoperability between sources that use different versions of an ontology.

Before we define backward-compatibility, we will first characterize and compare different types of revisions using the formal model developed in Section 3.3. 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. Since the revision only adds a sentence to the corresponding theory $T' \supset T$, and since first-order logic is monotonic any logical consequence of T is also a logical consequence of T'. Thus, when a revision that adds rules provides an alternate perspective of a legacy data source, 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 P semantically subsumes P'.

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 w.r.t. O that are not well-formed w.r.t. 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 when the symbols are removed the sentences are no longer well-formed. Thus, $T' \subseteq T$ and due to the monotonicity of definite 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 $T' \supset T$ 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 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. Thus, 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.

We can now define backward-compatibility: an ontology revision O' can be said to be backward-compatible with an ontology O if for any data source $S =< O, D_S >$, the perspective $P' =< S, O' >$ semantically subsumes the perspective $P =< S, O >$. Put simply, if every logical consequence of the original is also a consequence of the revision, then the revision is backward-compatible. By our analysis above, if a revision only adds categories, relations, or rules then it is backward compatible with the original, while if it removes any of these components then it is not backward compatible.

With this notion of backward compatibility, agents can assume with some degree of confidence that a perspective that uses the backward compatible revision will not alter the original meaning of the data source, but instead supplement it with information that was originally considered an implicit assumption of the ontology. Agents that don't wish to assume anything, may still access the original version because it still exists at the original URL. However, it should be noted that this versioning mechanism is dependent on the compliance of the ontology designers. Since an ontology is merely a file on a web server, there is nothing to prevent its author from making changes to an existing ontology version. This is the price we pay for have having a system that is flexible enough to cope with the needs of diverse user communities while being able to change rapidly. However, we presume that users will gravitate towards ontologies from sources that they can trust and ontologies that cannot be trusted will become obsolete.

Although, ideally integration in SHOE is a byproduct of ontology extension, a distributed environment in which ontologies are rapidly changing is not always conducive to this. Even when ontology designers have the best intentions, a very specialized concept may be simultaneously defined by two new ontologies. To handle such situations, periodic ontology integration must occur. Ontologies can be integrated using a new ontology that maps the related concepts using inference rules, by revising the relevant ontologies to map to each other, or by creating a new more general ontology which defines the common concepts, and revising the relevant ontologies to extend the new ontology. We discuss each of these solutions in more detail in [21].

3.4.3 Scalability

The scalability of a knowledge representation depends on the computational complexity of the inferences that it sanctions. We intentionally omitted from SHOE features such as disjunction and negation that typically make knowledge representation systems intractable. Since SHOE is essentially a set of Horn clauses, a naive forward-chaining inference algorithm can be executed in polynomial time and space in the worst case. Of course, the expected size of an extensional database built from the Web makes this an undesirable option.

Fortunately, SHOE can be mapped to datalog, and take advantage of optimized algorithms such as magic sets [40]. The semantics of SHOE categories can be easily described using a datalog rule. For example, category membership such as the fact that a Person is a Mammal may be expressed by using unary predicates and a rule of the form:

```
Mammal(X) :- Person(x)
```

Since basic data types are only used to determine the well-formedness of SHOE instances, and to determine what kind of comparison to perform, they do not need a datalog equivalent. The axioms generated for relations are simple Horn clauses, and can be expressed easily. Since SHOE's inferential rules are basically Horn clauses, they also map directly to datalog. Furthermore, SHOE's limited variable rule ensures that all SHOE inference rules are safe. Finally, datalog is assumed to have a set of built-in predicates that can be used for the comparison clauses. Thus, SHOE is equivalent to safe datalog without negation.

Obviously, SHOE systems can benefit from the datalog research, but the massive size of the resulting KBs may still yield unacceptable performance. Therefore SHOE has a modular design that allows systems to cleanly provide differing degrees of inferential capability. For example, a system may chose only to implement transitive category membership, or may chose to implement no inference at all, thus providing only access to the extensional database. Although such systems might be incomplete reasoners with respect to the intended

perspective (see section 3.3), they can be complete reasoners with respect to an alternate perspective based on an ontology that is identical to that specified by the data source with the exception that it omits all inference rules and/or category definitions. In large data settings, data that can be inferred from one source may be explicit in another, narrowing the gap between the capabilities of complete and incomplete reasoners.

4 Implementation Issues

In the previous section we described the semantics of SHOE. In this section we consider the design of systems that incorporate SHOE and discuss the features and components that are required of such systems. We want to emphasize that SHOE is a language and a philosophy; it does not require any particular method of implementation.

We begin by discussing implementation of query engines that support SHOE's semantics in an efficient way. This is necessary for any system that wants to process significant amounts of SHOE information. We then discuss issues related to building a complete SHOE system that supports the design and use of ontologies, markup of web pages with semantics, and use of this information by agents or query tools.

4.1 Efficient SHOE Query Engines

In order to deal with large amounts of SHOE information, the design or selection of the back-end SHOE engine is very important. While of course SHOE can be implemented relatively easily in semantically sophisticated knowledge representation systems like LOOM or CYC-L, the language is intended to be feasibly implementable on top of fast, efficient KR systems with correspondingly simpler semantics. In order for a SHOE system to support all possible intended perspectives, it must be able to handle at least the following features:

- n-ary predicates
- inference of category membership
- constrained horn-clause inference
- built-in comparison predicates such as $=$, $<$, and $>$

SHOE is also intended to be modular in design: an agent might implement all of SHOE except the inferential rules, for example, depending on domain need.

We understand that in the worst case implementing even the simple semantics described above can be tricky. However, SHOE's goal is to balance semantic expressivity with computational complexity *in practice* given reasonable assumptions about the nature of distributed domain like the World Wide Web. A particular assumption is that while the World Wide Web potentially contains a great deal of *data*, and a great many distributed SHOE ontologies may result in a large ruleset, nonetheless in general the cyclic dependencies in these rules will be relatively highly localized to a given ontology's domain. In our experience the distributed nature of SHOE ontologies tends to promote a modular ontology design, and the hierarchical nature of SHOE ontologies tends to result in acyclic, feed-forward intra-ontology rule dependencies.

In the following sections we discuss how to implement SHOE, or parts of SHOE, using three efficient systems with different reasoning capabilities: XSB [38], an open source, logic programming system that can be used as an deductive database engine, Parka [17, 39], a high-performance KR system whose roots lie in semantic networks and frame systems, and a generic relational database management system.

4.1.1 XSB

XSB is a logic programming and deductive database system developed at SUNY Stony Brook. XSB is actually much more expressive than datalog, and thus can be used to completely implement SHOE. The transformation of SHOE into an XSB program mirrors the construction of the set of axioms for an ontology as defined in Section 3.3. First, some form of renaming must be used to ensure that names used in different ontologies are distinct. This can be accomplished by simply appending an ontology unique string to each name. Then for each parent category P specified for some category C, we have a statement of the form:

```
P(x) :- C(X).
```

and for each inference rule we similarly have it's Prolog equivalent, although SHOE inference rules with n clauses in the <IF-THEN> section will actually have n statements, one with each clause at the head. Each category or relation assertion made by an instance adds a ground atom to the program. If we wish to attach sources to assertions then we must add an additional argument to each of the atoms to indicate its source.

In practice a compiled XSB program representing some set of SHOE data loads quickly and will return answers to queries in a reasonable amount of time. For example, a 50,000 line program loads in 20 seconds and answers many common queries in less than a second. However, since the information available to a web agent will be ever-changing, we cannot expect to have pre-compiled programs for many applications.

4.1.2 Parka

Parka is a high-performance knowledge base system developed at the University of Maryland, and is capable of performing complex queries over very large knowledge bases in less than a second [39]. For example, when used with a Unified Medical Language System (UMLS) knowledge base consisting of almost 2 million assertions, Parka was able to execute typical recognition queries in under 2 seconds. In practice, a Parka knowledge base loads more quickly than an equivalent XSB program and the system has a faster query response time. Like SHOE, Parka uses categories, instances, and n-ary predicates. Parka's basic inference mechanisms are transitive category membership and inheritance.

Parka includes a built-in subcategory relation between categories (isa) and a categorization relation between a category and an instance (instanceof). Parka also includes a predicate for partial string matching, and a number of comparison predicates. One of Parka's strong suits is that it can work on top of SQL relational database systems, taking advantage of their transaction guarantees while still performing very fast queries.

Parka can support many of SHOE's semantics directly; SHOE's subcategory inference maps to Parka's isa relation, and SHOE's categorization of instances maps to Parka's instanceof relation. Parka does not have general inferential capabilities, and thus cannot support SHOE's inference rules. However, its ability to perform categorization and generalization over large taxonomies supports the most common type of SHOE inference. If additional SHOE inference rules are important to an agent, they must be implemented using an inference engine that interfaces with the Parka KB, although how to do this efficiently is an open research question.

4.1.3 Relational Database Management Systems

Lastly, we consider commercial relational database management systems (RDBMSs) because they have been designed to efficiently answer queries over large databases. However, this efficiency comes at a cost: there is no way to explicitly specify inference.

Still, mapping of much of SHOE is possible in an RDBMS. Each n-ary SHOE relation can be represented by a database relation with n attributes. Categories can be represented by a binary relation isa. Even certain types of inference rules can be implmented in RDBMSs. As described by Ullman [40], for any set of safe, non-recursive Datalog rules with stratified negation, there exists an expression in relational algebra that computes a relation for each predicate in the set of rules. Although the semantics of SHOE are safe and include no negation, SHOE rules can be recursive and therefore, some but not all, of the rules could be implemented using views. Additionally, some commercial RDBMSs include operators to compute the transitive closure of a relation (e.g., the CONNECT WITH option in the Oracle SELECT operator). More complex dependencies must either be ignored or implemented in a procedural host language.

4.2 System Design Issues

There are a number of choices that must be made in designing a SHOE system. These choices can be divided into the categories of ontology design, annotation, accessing information, and information processing. In this section, we provide an overview of each of these areas.

4.2.1 Ontology Design

Ontology design can be a time consuming process and is the subject of extensive research which is beyond the scope of this paper. It is believed that an ontology designer can save time and increase interoperability with other domains by identifying a set of existing ontologies that can be extended for his use. To assist the designer, there should be a central repository of ontologies. A simple repository could be a set of web pages that categorize ontologies, while a more complex repository may associate a number of characteristics with each ontology so that specific searches can be issued. A web-based system that uses the later approach is described in [41].

Another aspect of ontology creation is the availability of the ontology. Internet delays, especially over long distances, can result in slow downloads of ontologies. To remedy this, commonly used ontologies can be mirrored by many sites. To use the most accessible copy of an ontology, users should be able to specify preferred locations for accessing particular ontologies. In this way, the URL field in the <USE-ONTOLOGY> tag is only meant as a recommendation.

Some ontologies may be proprietary and thus placing them on the Web is undesirable. Such ontologies could be maintained on an intranet, assuming that is where the annotated information is stored too. In general, if the SHOE instances that use an ontology are available to a user, the ontology should also be available, so that SHOE-enabled software can appropriately interpret the instances.

4.2.2 Annotation

Annotation is the process of adding SHOE semantic tags to web pages. This can be the biggest bottleneck in making SHOE available. How to go about this process depends on the domain and the resources available. If no SHOE tools are available, then annotations can be made using a simple text editor. However, this requires familiarity with the SHOE specification and is prone to error. Therefore, we have provided the Knowledge Annotator, a graphical tool that allows users to add annotations by choosing items from lists and filling in forms. The Knowledge Annotator is described in more detail in Section 4.4.3.

If there are many many pages to annotate, it may be useful to find other methods to insert SHOE into them. Large organizations that produce data on a regular basis often create web pages from the content of databases using scripts. In such situations, these scripts can be easily modified to include SHOE tags. In other cases, there may be a regular format to the data, and a short program can be written to extract relations and categories based on pattern matching techniques. Finally, NLP techniques have had success in narrow domains, and if an appropriate tool exists that works on the document collection, then it can be used to create statements that can be translated to SHOE. It should be mentioned that even if such an NLP tool is available, it is advantageous to annotate the documents with SHOE because this gives humans the opportunity to correct mistakes and allows query systems to use the information without having to reparse the text.

4.3 Accessing Information

Once SHOE ontologies and instances are available on the Web, SHOE agents and search engines must be able to access this information. There are two basic approaches: direct access and repository-based access. In the direct access approach, the software makes an HTTP request to the relevant web page or pages and extracts the SHOE markup. The advantage of this approach is that extracted knowledge is guaranteed to be current. However, the latency in internet connections means that this approach cannot be realistically used in situations where many pages must be searched. Therefore, it is best used to respond to specific, localized queries, where incomplete answers are expected. It may also be used to supplement ordinary browsing with additional semantic information about pages in the neighborhood of a selected page.

The repository-based access approach relies on a web-crawler to gather SHOE information and cache it in a central location, which is similar to the way contemporary search engines work. Certain constraints may be placed on such a system, such as to only visit certain hosts, only collect information regarding a particular ontology, or to answer a specific query. Queries are then issued to the repository, rather than the the Web at large. The chief advantage of the this approach is that accessing a local KB is much faster than loading web pages, and thus a complete search can be accomplished in less time. However, since a web-crawler can only process information so quickly, there is a tradeoff between coverage of the Web and freshness of the data.

If the system revisits pages frequently, then there is less time for discovering new pages. Exposé, which is discussed in Section 4.4.4, is a SHOE web-crawler that enables the repository-based access approach.

4.3.1 Information Processing

Ultimately, the goal of a SHOE system is to process the data in some way. This information may be used by an intelligent web agent in the course of performing its tasks or it may be used to help a user locate useful documents. In the later case, the system may either respond to a direct query or the user may create a standing query that the system responds to periodically with information based on its gathering efforts.

As discussed in Section 4.1, this processing will need a component that stores the knowledge that has been discovered and allows queries to be issued against that knowledge. This tool should have an API that allows various user interfaces, both general and domain specific, to use the knowledge.

4.4 Existing Tools

To support the implementation of SHOE we have developed a number of general purpose tools. These tools are coded in Java and thus allow the development of platform independent applications and applets that can be deployed over the Web. Since Java is an object-oriented language, we will describe the design of these tools using object-oriented terminology. Specifically, we will use the term *class* to refer to a collection of data and methods that operate on that data and *object* to refer to an instantiation of a class. Additionally, the term *package* is used to identify a collection of Java classes.

4.4.1 The SHOE Library

The SHOE library is a Java package that can be used by other programs to parse files containing SHOE, write SHOE to files, and perform simple manipulations on various elements of the language. The emphasis is on KB independence, although these classes can easily be used with a KB API to store the SHOE information in a KB. The central class is one that represents a SHOE document and can be initialized using a file or internet resource. The document is stored in such a way that the structure and the format is preserved, while efficient access to and updating of the SHOE tags within the document is still possible. A SHOE document object may contain many instance or ontology objects. Since SHOE is order independent but the interpretation of some tags may depend on others in the same document, the document must be validated in two steps. The first step ensures that it is syntactically correct and creates the appropriate structures for each document component. The second step ensures that the SHOE structures are internally consistent with themselves and with the ontologies that they depend on.

In addition to having classes for instances and ontologies, there are classes that correspond to each of the other SHOE tags. These classes all have a common ancestor and include methods for reading and interpreting tags contained within them, modifying properties or components and validating that the object is consistent with the rules of the language. Each class uses data structures and methods that are optimized for the most common accesses to it. For example, the ontology class includes a hash table for quick reference to its elements by name. It also keeps track of the most abstract categories that it defines so these can be used as root nodes for trees that describe the taxonomic structure of the ontology.

An ontology manager class is used to cache ontologies. This is important because ontology information is used frequently and it is more efficient to access this information from memory than to access it from disk, or even worse, the Web. However, there may be too many ontologies to store them all in memory, and therefore a cache is appropriate. One of the most important features of this class is a method which resolves prefixed names. In other words, it determines precisely which ontology element is being referred to. This is non-trivial because prefix chains can result in lookups in a series of ontologies and objects can be renamed in certain ontologies. When objects that contain such prefixed names are validated, the names are resolved into an identifier that consists of the id and version of the ontology that originated the object and the name of the object within that ontology. This identifier is stored within the object to prevent unnecessary repetition of the prefix resolution process.

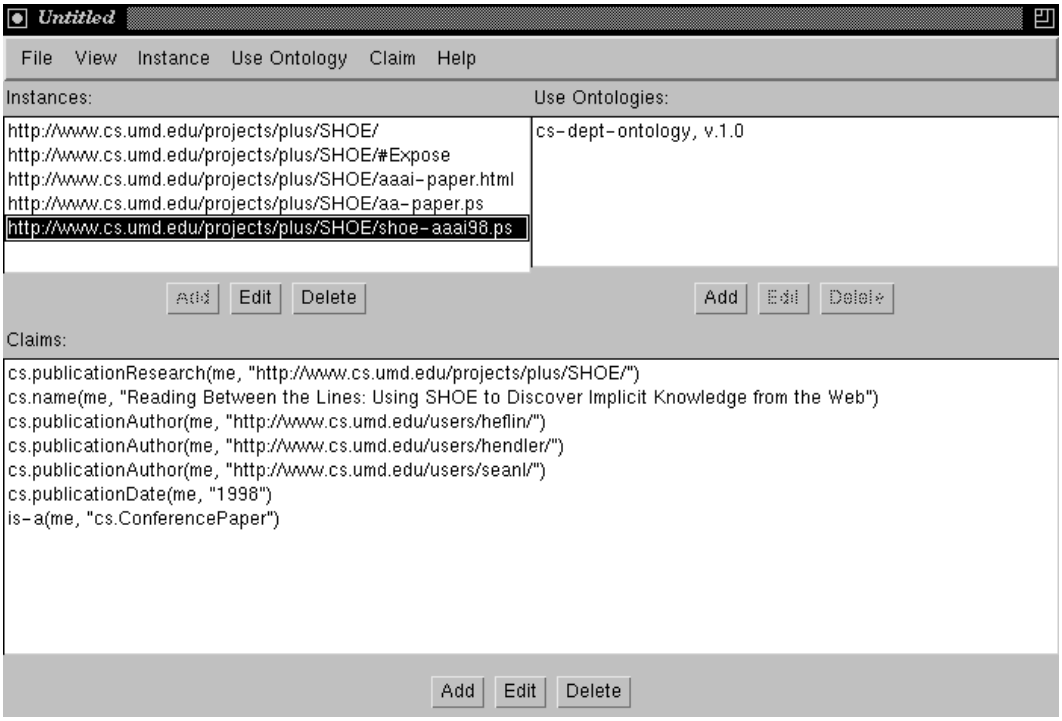

Figure 5: The Knowledge Annotator

4.4.2 The SHOE KB Library

The SHOE KB library is a Java package that provides a generic API for storing SHOE data and accessing a query engine. Applications that use this API can be easily modified to use a different reasoning system, thus allowing them to execute in a difference portion of the completeness / execution time tradeoff space. We have currently implemented versions of this API for Parka, XSB, and OKBC [11].

ShoeKb, the main class of the SHOE KB Library API contains methods for storing ontologies, storing SHOE document data and issuing queries. This class is supported by a KBInterface class which is loosely based on a restricted version of OKBC. Additional supporting classes include Sentence, Atom, Query, and Axiom objects.

In SHOE's formal semantics, we stated that if an instance appears in an argument of a relation which is not of the base type, then we automatically infer that the instance is of the required type, rather than perform type checking. This can result in the addition of a large number of rules to the KB. To avoid this, we treat such situations as if the source has made an implicit category assertion: we store an assertion that the instance is of the required category.

4.4.3 Knowledge Annotator

The Knowledge Annotator is a tool that makes it easy to add SHOE knowledge to web pages by making selections and filling in forms. As can be seen in Figure 5, the tool has an interface that displays instances, ontologies, and claims. Users can add, edit or remove any of these objects. When creating a new object, users are prompted for the necessary information. In the case of claims, a user can choose the source ontology from a list, and then choose categories or relations from a corresponding list. By default, the available relations will automatically filter based upon whether the instances entered are known to be of the correct types for the argument positions. Of course, since SHOE will infer a type from the use of an instance in a relation, this can be overriden. 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[6] and converts the inputs into legal SHOE syntax. For these reasons, only a rudimentary understanding of SHOE is necessary to markup web pages.

4.4.4 Exposé

After SHOE content has been created, whether by the Knowledge Annotator or other tools, it can be accessed by Exposé, a web-crawler that searches for web pages with SHOE markup. Exposé stores the knowledge it gathers in a knowledge base, and thus can be used as part of a repository-based system. The web-crawler is initialized by specifying a starting URL, a repository, and a set of constraints on which web sites or directories it may visit. These constraints allow the search to focus on sources of information that are known to be of high quality and can be used to keep the agent from accumulating more information than the knowledge base can handle. Exposé can either build a new repository of SHOE information or revisit a set of web pages to refresh an existing repository.

A web-crawler essentially performs a graph traversal where the nodes are web pages and the arcs are the hypertext links between them. Exposé maintains an open list of URLs to visit, and a closed list of URLs that have already been visited. When visiting web pages, it follows standard web robot etiquette by not requesting pages that have been disallowed by a server's robot.txt file and by waiting 30 seconds between page requests, so as not to overload a server.

Upon discovering a new URL, Exposé 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 using the SHOE library, identifies all of the hypertext links, category instances, and relation arguments within the page, and evaluates each new URL as above. Finally, the agent uses the SHOE KB Library API to store SHOE category and relation assertion in a specified knowledge base.

4.4.5 SHOE Search

SHOE Search [22] is a tool used to query the information that had been loaded into a SHOE KB. This interface, which is shown in Figure 6, gives users a new way to browse the web by allowing them to submit structured queries and open documents by clicking on the URLs in the results. The user first chooses an ontology against which the query should be issued and then chooses the class of the desired object from a hierarchical list. After the system presents a list of all properties that could apply to that object, and the user has typed in desired values for one or more of these properties, the user issues a query and is presented with a set of results in a tabular form. If the user double-clicks on a binding that is a URL, then the corresponding web page will be opened in a new window of the user's web browser.

SHOE Search is primarily used as a Java applet, and as such is executed on the machine of each user who opens it. This client application communicates with a central knowledge base through a server that is located on the website that hosts the applet. When a user starts the applet, it sends a message to the server. The server responds by creating a new process and establishing a socket for communication with the applet. When the user issues a query, it is sent to the server, which processes it and sends the answers back to the applet. SHOE Search uses the SHOE KB Library and consequently supports the use of Parka, XSB, or OKBC compliant knowledge bases as a backend, although it can be tailored for use with other knowledge representation systems.

5 How SHOE Can Be Used: Two Applications

We have prototyped two applications to demonstrate the usefulness and capabilities of SHOE. The first was an internal application to initially validate the concept and the second was developed for an outside party

[6] Here correctness is in respect to SHOE's syntax and semantics. The Knowledge Annotator cannot verify if the user's inputs properly describe the page.

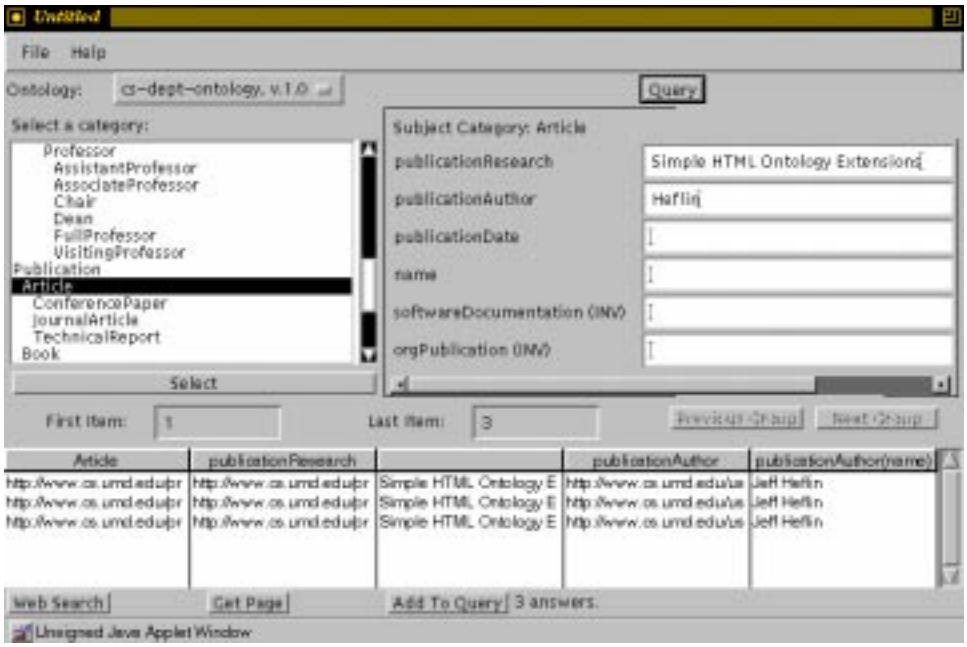

Figure 6: SHOE Search

to solve a real-world problem. We describe each of these applications and then provide some lessons learned from their development.

5.1 A Computer Science Department Application

The first application was developed as a proof of concept of the language. We chose the domain of computer science departments because it is simple and familiar to researchers interested in internet technology. We wanted to evaluate the ease of adding SHOE to web pages, the types of queries that could be constructed, and the performance of SHOE queries in a simple environment. The basic architecture, as shown in Figure 7, consists of annotating web pages using the Knowledge Annotator or some other tool, using Exposé to discover knowledge, and using a graphical Java applet to query the knowledge base that stores the knowledge.

The first step was to create a simple computer science department ontology that extends the base SHOE ontology. Some of the categories defined were Student, Faculty, Course, Department, Publication and Research. Relations such as publicationAuthor, emailAddress, advisor, and member were also defined. The final ontology[7] had 43 categories and 25 relations and was created by hand. It includes a standard HTML section to present a human readable description as well as a section with the SHOE syntax. In this way, the file serves the purpose of educating users in addition to providing machine understandable semantics.

The next step was to annotate a set of web pages (i.e., add SHOE semantic markup to them). Every member of the Parallel Understanding Systems (PLUS) Group marked up their own web pages. Although most members used the Knowledge Annotator, a few added the tags using their favorite text editors.

To get even more SHOE information about computer science departments, we looked for web pages with semi-regular structure. Most departments had lists of faculty, users, courses and research groups which fit this criteria. We extracted SHOE tags from this data by specifying patterns in the HTML source that marked the beginning and ends of instances that participated in relations or could be categorized according to our ontology. The resulting tags were added to the summary pages mentioned above. We also created a tool that could extract publication information from CiteSeer (*http://citeseer.nj.nec.com/cs*), an index of on-line computer science publications.

Exposé was used to acquire the SHOE knowledge from the web pages. This resulted in a total of

[7] This ontology is located at *http://www.cs.umd.edu/projects/plus/SHOE/onts/cs1.0.html*

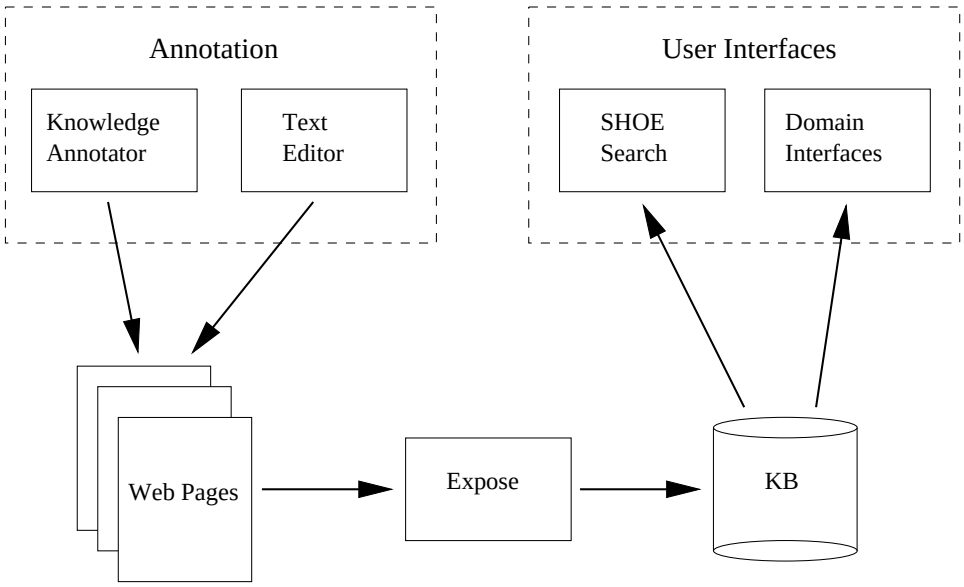

Figure 7: The SHOE Architecture

38,159 assertions, which were then stored in Parka. Although the KB for this demo is very small when compared to the scale of the entire Web, the initial results are promising. For example, a query of the form member(http://www.cs.umd.edu, x) $\wedge$ instance(Faculty, x) takes less than 250 milliseconds to answer.

Although Parka possesses most of the features required for a system to implement the full SHOE semantics, as mentioned in Section 4.1.2, it still lacks the ability to do arbitrary inference. Therefore, we also used Exposé to load an XSB database with the same data and perform queries. Queries to XSB were slower than those sent to Parka, but the response time was still generally in the range of a few seconds. Future work will involve a thorough performance evaluation using much larger knowledge bases and an evaluation of what percentage of the complete results are returned by Parka for typical real-world queries.

As this example demonstrates, there are many possible means of acquiring SHOE information. Here, text editors, a GUI, and custom pattern extraction tools were all used to encode the knowledge. This allowed the development of a significant number of SHOE assertions in days. By combining the ability to annotate one's own pages with the ability to make assertions about the content of other web pages, SHOE allows the efforts of information providers and professional indexers to be combined.

The possible benefits of a system such as this one are numerous. A prospective student could use it to inquire about universities that offered a particular class or performed research in certain areas. Or a researcher could design an agent to search for articles on a particular subject, whose authors are members of a particular set of institutions, and were published during some desired time interval. Additionally, SHOE can combine the information contained in multiple sources to answer a single query. For example, to answer the query "Find all papers about ontologies written by authors who are faculty members at public universities in the state of Maryland" one would need information from university home pages, faculty listing pages, and publication pages for individual faculty members. Such a query would be impossible for current search engines because they rank each page based upon how many of the query terms it contains.

5.2 A Food Safety Application

The Joint Institute for Food Safety and Applied Nutrition (JIFSAN), a partnership between the Food and Drug Administration (FDA) and the University of Maryland, is working to expand the knowledge and resources available to support risk analysis in the food safety area. One of their goals is to develop a website that will serve as a clearinghouse of information about food safety risks. This website must serve a diverse group of users, including researchers, policy makers, risk assessors, and the general public, and thus must

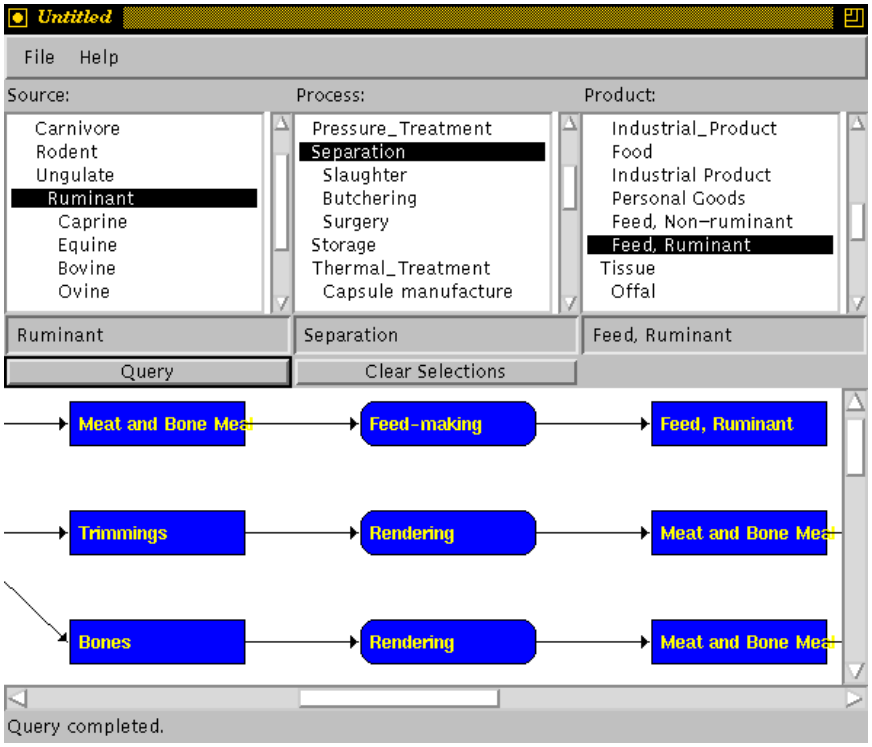

Figure 8: The TSE Path Analyzer

be able to respond to queries where terminology, complexity and specificity may vary greatly. This is not possible with keyword based indices, but can be achieved using SHOE. This section discusses our experiences using SHOE to support the TSE Risk Website, the first step in building a food safety clearinghouse.

In order to scope the project, JIFSAN decided to focus the SHOE effort on a specific issue of food safety. The chosen issue was Transmissible Spongiform Encephalopathies (TSEs), which are brain diseases that cause sponge-like abnormalities in brain cells. "Mad Cow Disease," which is technically known as Bovine Spongiform Encephalopathy (BSE), is the most notorious TSE, mainly because of its apparent link to Creutzfeldt-Jakob disease (CJD) in humans. Recent Mad Cow Disease epidemics and concerns about the risks BSE poses to humans continue to spawn international interest on the topic.

The initial TSE ontology was fleshed out in a series of meetings that included members of the FDA and the Maryland Veterinarian School. The ontology focused on the three main concerns for TSE Risks: source material, processing, and end-product use. Currently, the ontology has 73 categories and 88 relations.[8] In addition to specific TSE concepts such as Disease and Risk, general terms such as Person, Organization, Process, Event, and Location were defined. Twelve of the relations have three or more arguments, indicating the usefulness of n-ary relations. One reason for the need of n-ary relations is that scientific data tends to have many parameters. For example, the infectivityTitre relation measures the degree of infectivity in a tissue given a disease, source animal, and tissue type.

Following the creation of the initial ontology, the team annotated web pages. There are two types of pages that this system uses. Since the Web currently has little information on animal material processing, we created a set of pages describing many important source materials, processes and products. The second set of pages are existing TSE pages that provide general descriptions of the disease, make recommendations or regulations, and present experimental results. Early annotations were difficult because the original ontology did not have all of the concepts that were needed.

When the initial set of pages was completed, we ran Exposé, using Parka as the knowledge base system. Since the TSE Ontology currently does not define inference rules, Parka is able to provide a complete

[8] Those interested in the details of the ontology can view it at *http://www.cs.umd.edu/projects/plus/SHOE/onts/tseont.html*

reasoning capability for it. The Parka KB can be queried using SHOE Search as discussed earlier, but JIFSAN also wanted a special purpose tool to help users visualize and understand the processing of animal materials.

To accommodate this, we built the TSE Path Analyzer, a graphical tool that can be used to analyze how source materials end up in products that are eventually consumed by humans or animals. This information is extremely valuable when trying to determine the risk of contamination given the chance that a source material is contaminated. It is expected that information on each step in the process will be provided on different web sites (since many steps are performed by different companies), thus using a language like SHOE is essential to integrating this information. The TSE Path Analyzer allows the user to pick a source, process and/or end product from lists that are derived from the taxonomies of the ontology. The system then displays all possible pathways that match the query; an example is shown in Figure 8. Since these displays are created dynamically based on the semantic information in the SHOE web pages, they are kept current automatically, even when the SHOE information on some remote site is changed.

We are still testing the system and gradually accumulating the mass of annotated web pages that is necessary to make it really useful. When it is publicly released, the system's operation will be as follows:

1. Knowledge providers who wish to make material available to the TSE Risk Website use the Knowledge Annotator or other tools to add SHOE markup their pages. The instances within these pages are described using elements from the TSE Ontology.

2. The knowledge providers then place the pages on the Web and notify JIFSAN.

3. JIFSAN reviews the site and if it meets their standards, adds it to the list of sites that Exposé, the SHOE web-crawler, is allowed to visit.

4. Exposé crawls along the selected sites, searching for more SHOE annotated pages with relevant TSE information. It will also look for updates to pages.

5. SHOE knowledge discovered by Exposé is loaded into a Parka knowledge base. Alternate KBs may be added later.

6. Java applets on the TSE Risk Website access the knowledge base to respond to users' queries or update displays. These applets include the TSE Path Analyzer and SHOE Search.

It is important to note that new websites with TSE information will be forced to register with JIFSAN. This makes Exposé's search more productive and allows JIFSAN to maintain a level of quality over the data they present from their website. However, this does not restrict the ability of approved sites to get current information indexed. Once a site is registered, it is considered trusted and Exposé will revisit it periodically.

5.3 Lessons Learned

While formally evaluating the methodologies and processes described above is difficult, the process has taught us some valuable lessons.

The knowledge engineering aspects of the TSE application were much more difficult than those of CS department application. We believe the main reason for this is that the problem had ill-defined boundaries. In an attempt to describe an ontology to the best detail possible, it is easy to lose sight of the original intent of the ontology. As a result, we have developed the following guidelines for determining the scope of the ontology:

- What kind of pages will be annotated?
- What sorts of queries can the pages be used to answer?
- Who will be the users of the pages?
- What kinds of objects are of interest to these users?
- What are the interesting relationships between these objects?

During the annotation process, we found it helpful to have guidelines for identifying what concepts to annotate. Based on our experiences with this application, we make the following suggestions. First, if the document represents or describes a real world object then an instance whose key is the document's URL should be created. Second, hyperlinks are often signs that there is some relation between the object in the document and another object represented by the hyperlinked URL. If a hyperlinked document does not have SHOE annotations, it may also be useful to make assertions about its object. Third, one can create an instance for every proper noun, although in large documents this may be excessive. If these concepts have a web presence, then the corresponding URLs should be used as keys, otherwise a unique key can be created by appending a "#" and a unique string to the end of the annotated document's URL.

However, even with these guidelines, the annotation process was more difficult for the TSE application. Although this is partly because no annotators were both AI researchers and domain experts, we believe the main difficulty was due to the difference in the data. In the computer science application, each instance of the major concepts (e.g., department, course, professor, publication) typically has its own home page. These pages often have a form or list-based look and feel that can easily be mapped into predicates or a frame system. The content of TSE pages on the other hand mostly consists of free text documents that refer to shared entities such as BSE or the North American continent. As such, determining the relevant relations required much more effort and choosing a single URL as a key was difficult. In the later case, we created constants in the ontology to represent the shared objects, although we are investigating better alternatives.

Annotators of TSE pages also had difficulty in determining the level of detail that is required. Although more detailed information increases the likelihood that a page will be returned to a precise query that matches its contents, adding such annotations is a time consuming process and certain details are likely to never be of use to anyone. The following guidelines can be helpful in making these decisions:

- The utility of a page can be considered from a search perspective. If some of the information consumers are available, they can be asked to make a list of the most important questions that the page can be used to answer. The information provider can translate these questions into SHOE queries and make sure that the categories and relations in those queries are correctly specified in the page.
- A summary of the document should contain the important concepts of the document. If a summary is available, it helps the knowledge provider to clearly understand the value of the document.
- If the document author is available, he or she can be asked to identify the key or novel statements made by the document. Comparison of the nouns and verbs in these statements to the ontology can be used to discover useful assertions.

Besides learning the need for annotation and ontology design guidelines, we learned that web users are often willing to sacrifice power for simplicity. When we demoed SHOE at a TSE conference, we found that most users were much more impressed with the Path Analyzer than with tools like SHOE Search that allowed a wider range of possible queries. They cited the fact that it was easier to learn and that it displayed the results in a customized way that allowed them to explore the problems of interest to them.

6 Future Work

So far we have described the requirements for an internet knowledge representation language, described one such language and then analyzed the language by using it in various applications. This initial work shows much promise, but there is still much work to do. Future work can be divided into three areas of research: addressing performance issues, enhancing functionality of the language, and improving the usability of the language and tools. Each of these areas is discussed in one of the following subsections.

6.1 Performance

One of the most critical components in a SHOE system is the underlying knowledge representation system. To deal with the quantity of information available on the Web, such a system must strike a balance between scalability and complete reasoning. As we discussed in Section 4.1, there are many possible candidate back-end systems. For example, a deductive database can support the full semantics of SHOE, but there

will be performance tradeoffs. On the other hand, relational database management systems can provide performance gains but at the cost of no inference. We envision a Web in which SHOE search engines differentiate themselves by the degree of inference supported and the size of the underlying KB (which depends on whether the engine is general purpose or domain specific). Each of these systems can be thought of as providing an alternate perspective for the data sources, and the choice of which is best depends on the priorities of the user. To help in such decisions, we will measure the costs and benefits of different types of KR systems by varying the size of the KB, the complexity of the inference rules, and the types of queries.

Due to the changing nature of the Web, another important performance issue is the freshness of the data. When a web-crawler is used as we have done in our demonstrations, the data is only as current as the last time the crawler was able to visit the pages. While this may be sufficient for a basic search application, it could be problematic for a comparison shopping agent. In the later case, an on-line search agent would be more applicable, but difficult issues must be addressed to make such agents work. For example, the agent must be able to choose a good starting point for a particular search and then must use its knowledge to help it traverse the Web and locate the requested information efficiently.

6.2 Functionality

Although we have initially restricted the SHOE language to reduce computational complexity and make it easier to learn, our experiences have led us to consider a number of enhancements. In particular, additional features may further address the issues of trust in sources, inconsistency and interoperability.

Due to reliability considerations of a source and potential inconsistencies between sources, the issue of trust becomes important. Trust can be very subjective, and two individuals may disagree on whether a particular source is reliable. Since the formal model described in Section 3.3 defines a perspective as a data source viewed in the context of an ontology, a user could create a perspective that restricts the data source to those web pages which are considered reliable by him or her. However, this is a simplistic notion of trust because a source may be reliable only on certain subject matter, reliability may depend on supporting evidence, or differing degrees of belief could be considered. A solution to the first two problems is to create special ontologies that provide belief systems using sets of rules that filter out assertions which may be suspect. Such ontologies will require special SHOE relations such as $claims(x, c)$, which is true if x is the source of claim c, and $believe(c)$, which is true if the agent should believe claim c. On the other hand, the varying degrees of belief could be accommodated by allowing a user to define a special function for propagating belief from antecedents to consequents.

Although SHOE avoids contradictions by disallowing logical negation and constraints such as single-valued relations, such constructs are very useful in validating and evaluating data that is discovered. For this reason we are considering the addition of some limited constraints to the language, but in a distributed system a constraint may only apply from a particular perspective or the violation of a constraint may be due to bad data that was discovered earlier. As such, a constraint should not prevent data from being included in the KB, instead it should be used as a filter at query time that results in a warning or lowering of the belief in a particular claim. For example, the more constraints that some datum violates, the less likely that we would be to believe it.

Due to the distributed nature of SHOE ontologies, the need to convert between different measurement units is inevitable. One way to achieve this is to allow arithmetic functions in inference rules. However, if an arithmetic function is used recursively in a rule, inference procedures may never terminate. Other types of conversions may also be necessary, and arbitrary functions should be considered. Since the definition of an arbitrary function would require a much more complex language, an alternative may be the specification of a remote procedure call.

Although, SHOE takes an open-world approach, there are many useful queries and actions that cannot be performed by web agents without closed-world information. Localized closed-world (LCW) statements [16] are a promising step in this direction. LCW statements can be used to state that the given source has all of the information on a given topic. LCW statements are more appropriate for the Web than the closed-world assumption, but there is still a question as to how a query system acquires the set of LCW statements that could be relevant. One possible extension to SHOE is to allow LCW statements to be expressed in the language.

6.3 Usability

Usability is too often overlooked, but is essential for success on the Web. Most users do not have the time or desire to learn complicated tools or languages. To evaluate the system's usability, we must ask questions such as: How easy is it to annotate pages? How long does it take to annotate a page? How user-friendly are the tools? Is the process intuitive?

The knowledge acquisition bottleneck has caused many knowledge based approaches to fail. SHOE hopes to overcome this problem by getting the largest possible number of individuals involved in the knowledge acquisition process by way of annotation. For such an approach to work, we must make the process simple and straightforward for the layperson. We are actively working with our users to determine what interfaces are the most intuitive. Certainly, the ultimate annotation process would be fully automatic, but due to limitations of NLP in general domains, this goal is currently unrealistic. However, a semi-automatic method that incorporated ideas from NLP or machine learning may simplify the the process for the user.

It is well known that ontology development is a difficult task. To keep up with the changing nature of the Web, it is important that good ontologies can be designed quickly. To this end, we intend to create a set of tools and methods to help in the design process. First, we plan to create an ontology design tool which is the ontology equivalent of the Knowledge Annotator. Second, we will design a library of SHOE ontologies, so that ontology authors can use SHOE's extension mechanism to reuse existing components and focus on the fundamental issues of their ontologies. To initialize our library, we can make use of publicly available ontologies such as those found on the Ontolingua Server [18]. We will write translation tools to and from the most common ontology formats. Third, we will try to identify how, if at all Web ontology design should differ from traditional ontology design. For example, we believe that SHOE ontologies will be used mostly to categorize information and find simple relationships. As such, extremely detailed ontologies may not be necessary.

7 Related Work

In recent years, there has been work to use ontologies to help machines process and understand Web documents. Fensel et al. [19] have developed Ontobroker, which proposes minor extensions to the common anchor tag in HTML. The theoretical basis for Ontobroker is frame logic, a superset of Horn logic that treats ontology objects as first class citizens. However, this approach depends on a centralized broker, and as a result, the web pages cannot specify that they reference a particular ontology, and agents from outside the community cannot discover the ontology information. Kent [26] has designed the Ontology Markup Language (OML) and the Conceptual Knowledge Markup Language (CKML), which were influenced by SHOE, but are based on the theories of formal concept analysis and information flow. However, the complexity of these theories make it unlikely that this language will be accepted by the majority of existing web developers and/or users. The Ontology Interchange Language (OIL) [14] is a new web ontology language that extends RDF and RDF Schema with description logic capabilities. More recently, the DARPA Agent Markup Langauge (DAML) [23] has attempted to combine the best features of SHOE, RDF, and OIL. Jannink et al. [25] suggest a different approach from creating web ontology languages and annotating pages; they propose 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.

Querying the Web is such an important problem that a diverse body of research has be directed towards it. Some projects focus on creating query languages for the Web [1, 27], but these approaches are limited to queries concerning the HTML structure of the document and the hypertext links. They also rely on index servers such as AltaVista or Lycos to search for words or phrases, and thus suffer from the limitations of keyword search. Work on semistructured databases [33] is of great significance to querying and processing XML, but the semistructured model suffers the same interoperability problems as XML. Even techniques such as data guides will be of little use when integrating information developed by different communities in different contexts. Another approach involves mediators (or wrappers), custom software that serves as an interface between middleware and a data source [42, 35, 37]. When applied to the Web, wrappers allow users to query a page's contents as if it was a database. However, the heterogeneity of the Web requires that a multitude of custom wrappers must be developed, and it is possible that important relationships cannot be extracted from the text based solely on the structure of the document. Semi-automatic generation of

wrappers [2] is a promising approach to overcoming the first problem, but is limited to data that has a recognizable structure.

In order to avoid the overhead of annotating pages or writing wrappers, some researchers have proposed machine learning techniques. Craven et al. [13] have trained a system to classify web pages and extract relations from them in accordance with a simple ontology. However, this approach is constrained by the time-consuming task of developing a training set and has difficulty in classifying certain kinds of pages due to the lack of similarities between pages in the same class.

8 Conclusion

In this paper, we have described many of the challenges that must be addressed by research on the Semantic Web and have described SHOE, one of the first languages to explictly address these problems. SHOE provides interoperability in distributed environments through the use of extensible, shared ontologies, the avoidance of contradictions, and localization of inference rules. It handles the changing nature of the Web with an ontology versioning scheme that supports backward-compatibility. It takes steps in the direction of scalability by limiting expressivity and allowing for different levels on inferential support. Finally, since the Web is an "open-world," SHOE does not allow conclusions to be drawn from lack of information.

To demonstrate SHOE's features, we have described applications that show the use of SHOE. We've developed a freely available ontology for computer science pages, and we've also worked with biological epidemiologists to design an ontology for a key food safety area. These applications show that SHOE can exist on the web, and that tools using SHOE can be built and used. Future work includes an evaluation of different implementation strategies, enhancements that make the language even more suit to distributed, dynamic environments, and the development of tools that make the language more user friendly.

Although we believe SHOE is good language that has practical use, we do not mean to suggest that it solves all of the problems of the Semantic Web. We are at the beginnings of a new and exciting research field and there is still much research to do. As early "pioneers," we hope that our experience with SHOE can inspire and inform others. A key goal of this project is to raise the issues that are crucial to the development of the Semantic Web and encourage others to explore them. To this end, we have made SHOE freely available on the Web, including the Java libraries and our prototype tools. Interested readers are urged to explore our web pages at *http://www.cs.umd.edu/projects/plus/SHOE/* for the full details of the language and the applications.

Acknowledgments

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

References

[1] G. Arocena, A. Mendelzon and G. Mihaila, Applications of a Web Query Language, in: *Proceedings of ACM PODS Conference* Tuscon, AZ (1997).

[2] N. Ashish and C. Knoblock, Semi-automatic Wrapper Generation for Internet Information Sources, in: *Proceedings of the Second IFCIS Conference on Cooperative Information Systems (CoopIS)* Charleston, SC (1997).

[3] T. Berners-Lee, and D. Connolly, Hypertext Markup Language - 2.0, IETF HTML Working Group, at: http://www.cs.tu-berlin.de/~ jutta/ht/draft-ietf-html-spec-01.html (1995).

[4] D. Bobrow and T. Winograd, An overview of KRL, a knowledge representation language, *Cognitive Science* 1(1) (1977).

[5] P. Borst, J. Benjaminm, B. Wielinga, and H. Akkermans. An Application of Ontology Construction, in: *Proceedings of ECAI'96 Workshop on Ontological Engineering* (1996).

[6] R. Brachman and J. Schmolze, An overview of the KL-ONE knowledge representation system, *Cognitive Science*, 9(2) (1985).

[7] R. Brachman, D. McGuinness, P.F. Patel-Schneider, L. Resnick,and A. Borgida, Living with Classic: When and how to use a KL-ONE-like language, in: J. Sowa, ed., *Explorations in the representation of knowledge* (Morgan-Kaufmann, CA, 1991).

[8] T. Bray, J. Paoli and C. Sperberg-McQueen, Extensible Markup Language (XML), W3C (World Wide Web Consortium), February 1998, at: http://www.w3.org/TR/1998/REC-xml-19980210.html. (1998)

[9] T. Bray, D. Hollander, and A. Layman, Namespaces in XML, W3C (World Wide Web Consortium), January 1999, at: http://www.w3.org/TR/1999/REC-xml-names-19990114/.

[10] D. Brickley and R.V. Guha, Resource Description Framework (RDF) Schema Specification (Candidate Recommendation), W3C (World-Wide Web Consortium) (2000). (At http://www.w3.org/TR/2000/CR-rdf-schema-20000327)

[11] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp and J. Rice, OKBC: A Programatic Foundation for Knowledge Base Interoperbility, in: *Proc. of AAAI-98* (AAAI/MIT Press, Menlo Park, CA, 1998) 600-607.

[12] J. Clark, XSL Transformations (XSLT) W3C (World-Wide Web Consortium) (1999). (At http://www.w3.org/TR/1999/REC-xslt-19991116)

[13] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigram, and S. Slattery, Learning to Extract Symbolic Knowledge From the World Wide Web, in: *Proceedings of the Fifteenth American Association for Artificial Intelligence Conference (AAAI-98)* (AAAI/MIT Press, 1998).

[14] S. Decker, D. Fensel , F. van Harmelen , I. Horrocks, S. Melnik , M. Klein, and J. Broekstra, Knowledge Representation on the Web, in: *Proceedings of the 2000 International Workshop on Description Logics (DL2000)* (Aachen, Germany, August 2000).

[15] S. Dobson and V. Burrill, Lightweight Databases, in: *Proceedings of the Third International World Wide Web Conference (special issue of Computer and ISDN Systems)* 27(6) (Elsevier Science, Amsterdam, 1995).

[16] O. Etzioni, K. Golden, and D. Weld, Sound and efficient close-world reasoning for planning, *Artif. Intell.* 89 (1997) 113-148.

[17] M. Evett, W. Andersen and J. Hendler, Providing Computational Effective Knowledge Representation via Massive Parallelism, in: L. Kanal, V. Kumar, H. Kitano, and C. Suttner, eds., *Parallel Processing for Artificial Intelligence*, (Elsevier Science, Amsterdam, 1993).

[18] A. Farquhar, R. Fikes and J. Rice The Ontolingua Server: A tool for collaborative ontology construction, *International Journal of Human-Computer Studies* 46(6) (1997) 707-727.

[19] D. Fensel, S. Decker, M. Erdmann, and R. Studer, Ontobroker: How to enable intelligent access to the WWW, in: *AI and Information Integration, Papers from the 1998 Workshop*, Technical Report WS-98-14 (AAAI Press, Menlo Park, CA, 1998).

[20] M. Grüninger. Designing and Evaluating Generic Ontologies, in *Proceedings of ECAI'96 Workshop on Ontological Engineering* (1996).

[21] J. Heflin, and J. Hendler, Dynamic Ontologies on the Web, in: *Proc. of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)* (AAAI/MIT Press, Menlo Park, CA, 2000) 443-449.

[22] J. Heflin and J. Hendler, Searching the Web with SHOE, in: *Artificial Intelligence for Web Search. Papers from the AAAI Workshop. WS-00-01* (AAAI Press, Menlo Park, CA, 2000) 35-40.

[23] J. Hendler and D. McGuinness. The DARPA Agent Markup Language, *IEEE Intelligent Systems* 15(6) (2000) 72-73.

[24] ISO (International Organization for Standardization) *ISO 8879:1986(E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML)* (International Organization for Standardization, Genevea, 1986).

[25] J. Jannink, S. Pichai, D. Verheijen, and G. Wiederhold., G. Encapsulation and Composition of Ontologies, in: *AI and Information Integration, Papers from the 1998 Workshop*, Technical Report WS-98-14 (AAAI Press, Menlo Park, CA, 1998) 43-50.

[26] R.E. Kent. Conceptual Knowledge Markup Language: The Central Core, in: *Twelfth Workshop on Knowledge Acquisition, Modeling and Management* (1999).

[27] D. Konopnicki and O. Shemueli, W3QS: A Query System for the World Wide Web, in: *Proceedings of the 21st International Conference on Very Large Databases* (Zurich, Switzerland, 1995).

[28] O. Lassila, Web Metadata: A Matter of Semantics, *IEEE Internet Computing* 2(4) (1998) 30-37.

[29] D. Lenat and R. Guha, *Building Large Knowledge Based Systems*, (Addison-Wesley, MA, 1990).

[30] S. Luke and J. Heflin, *SHOE 1.01, Proposed Specification*, at: http://www.cs.umd.edu/projects/plus/SHOE/spec.html (2000).

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

[32] R. MacGregor, The Evolving Technology of classification-based knowledge representation systems, in: J. Sowa, ed., *Explorations in the representation of knowledge* (Morgan-Kaufmann, CA, 1991).

[33] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, Lore: A Database Management System for Semistructured Data, *SIGMOD Record*, 26(3) (1997) 54-66.

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

[35] Y. Papakonstantinou, et al., A Query Translation Scheme for Rapid Implementation of Wrappers, in: *Procedeedings of the Conference on Deductive and Object-Oriented Databases(DOOD)* Singapore (1995).

[36] D. Ragget, Hypertext Markup Language Specification Version 3.0, W3C (World Wide Web Consortium), at: http://www.w3.org/pub/WWW/MarkUp/html3/CoverPage.html (1995).

[37] M. Roth, and P. Schwarz, Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources, in: *Proceedings of 23rd International Conference on Very Large Data Bases* (1997).

[38] K. Sagonas, T. Swift, and D. S. Warren, XSB as an Efficient Deductive Database Engine, in: R. T. Snodgrass and M. Winslett, editors, *Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94)* (1994) 442-453.

[39] K. Stoffel, M. Taylor and J. Hendler, Efficient Management of Very Large Ontologies, in: *Proceedings of American Association for Artificial Intelligence Conference (AAAI-97)* (AAAI/MIT Press, 1997).

[40] J. Ullman, Principles of Database and Knowledge-Base Systems, (Computer Science Press, MD, 1988).

[41] J. Vega, A. Gomez-Perez, A. Tello, and Helena Pinto, How to Find Suitable Ontologies Using an Ontology-Based WWW Broker, in: *International Work-Conference on Artificial and Natural Neural Networks, IWANN'99, Proceeding, Vol. II*, Alicante, Spain (1999) 725-739.

[42] G. Wiederhold, Mediators in the Architecture of Future Information Systems, *IEEE Computer* 25(3) (1992).

Appendix

A SHOE XML DTD

This appendix provides an XML DTD for SHOE.

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

<!-- Since this may be embeded in a document that doesn't have META
     elements, the SHOE version number is included as an attribute
     of the shoe element. -->
<!ATTLIST shoe
        version           CDATA        #REQUIRED >

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

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

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

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

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

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

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

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

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

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

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

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

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

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