SHOE: A Knowledge Representation Language for Internet Applications*

Jeff Heflin James Hendler
Sean Luke
Institute for Advanced Computer Studies, University of Maryland, College Park

Abstract

It is our contention that the World Wide Web poses challenges to knowledge representation systems that fundamentally change the way we should design KR languages. In this paper, we describe the Simple HTML Ontology Extensions (SHOE), a KR language which allows web pages to be annotated with semantics. We present a formalism for the language and discuss the features which make it well suited for the Web. We describe the syntax and semantics of this language, and discuss the differences from traditional KR systems that make it more suited to modern web applications. We also describe some generic tools for using the language and demonstrate its capabilities by describing two prototype systems that use it. We also discuss some future tools currently being developed for the language. 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

One of the venerable sub-fields of artificial intelligence is that of knowledge representation (KR). 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. Current KR meetings focus on the expressivity of description logics, elegant algorithms for terminological reasoning, and the use of advanced logics to push the expressivity of KR languages ever further. One traditional part of KR, again dating back to the very beginning, is the development of KR languages, the design of machine readable syntax with powerful formal semantics underlying it. As the applications for which KR is used have grown in sophistication, so too have the languages evolved ever more complex representational power. Early languages, such as KL-ONE [5] and KRL [4] have evolved into modern powerhouses like LOOM [20] , Classic [6], and CYC-L [19].

Underlying this evolution has been a dominant thread – the belief that expressivity is a critical property for a KR language. This assumption dates from the early days of KR, and has always been a major driver. The reason for this is clear. As Brachman and Schmolze stated in their seminal work on the KL-ONE language:

> KL-ONE is intended to represent general conceptual information and is typically used in the construction of the knowledge base of a single reasoning entity. A KL-ONE knowledge base can be thought of as representing the beliefs of the system using it ... KL-ONE provides a language for expressing an *explicit set of beliefs* for a rational agent.

they add a footnote saying

> Such a set of beliefs expressed in some representation language is what is typically meant by the term *knowledge base.*

This world view, in which the knowledge base represented the entire belief set of a complex reasoner, led to a need for languages which could provide deep relational information and represent highly expressive

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

inferences. This need for greater expressivity became one of the key qualities sought in designing advanced knowledge representation systems.

However, it is our contention that this fundamental philosophical underpinning of knowledge representation is brought into question in dealing with the Internet, and particularly the World Wide Web. We argue that the Internet fundamentally changes our view of what a knowledge base is, and in a corresponding manner what a KR language should be designed for! The World Wide Web itself should be viewed as a knowledge base: a massive source of information for agents to gather and make intelligent queries upon. However, such a knowledge base is fundamentally different from the sort of knowledge base described by Brachman and Schmolze, and provides a different set of motivations for those trying to develop KR languages that can harness the information potential of this important new knowledge-base resource.

Let's consider some of the key aspects of the Web when viewed as a knowledge base:

- **The Web is massive.** Recent estimates place the number of indexable web pages at about 550 million; hundreds of new pages are added every minute. If each page contained a *single* piece of agent-gatherable knowledge, the cumulative database would be large enough to bring virtually all existing knowledge representation systems to their knees. The reason is clear: most KR systems have semantics too rich to scale well. This is largely for a single reason: expressivity comes with the cost of increased computational complexity. Many KR languages have NP-hard or even Turing-complete complexity. For the Web, this is an unfathomable option - to scale to the size of the ever growing Web, KR systems must be far more efficient than they are today. This implies that KR languages for the web must perform in a different part of the expressivity/efficiency trade-off space.

- **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 AltaVista, currently the largest search engine, only has keyword indices for about 150 million (of the over 500 million) web pages in an index of about 200 gigabytes. However, in order to achieve more efficient reasoning, many KR systems make a closed-world assumption. That is, they assume that anything which is not entailed in the knowledge base is not true. But, 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.

  KR researchers on the web are very aware of this problem, and current research is trying to overcome it. For example, in [10], localized closed world (LCW) statements are used to improve query efficiency. 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. While this sort of solution helps, it is still difficult to use and quantify, hard to know when the assumptions hold, and it is still hard to maintain consistency given the dynamic nature of the changing web.

- **The Web is Dynamic.** The web changes at an incredible clip, far 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 others change in unpredictable intervals. Additionally, changes can vary in their significance. Obviously, the addition of punctuation, correction of spelling errors or reordering of a paragraph does not affect the semantic content of a document, while other changes may completely change meaning, or even remove large amounts of information (as when a web site is removed from the net). A KR system 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. A KR system's ontological framework must be flexible yet general to handle the Web's on-line economy of ideas.

To summarize, viewing the Web as the knowledge base changes the way we must look at KR and KR languages. Web systems 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. As authority on the Internet is distributed, it cannot and does not make any such promise. This lack of central control leads to a number of serious problems. Since there is often no editorial review or quality control of Web information, each page's *reliability* must be questioned. Since a web page that was useful one day can disappear the next, there is no guarantee on the *availability* of information. Since there are no integrity constraints on the Web, information from different sources can be in disagreement, leading to *inconsistency*. Some inconsistencies may be due to error, others due to philosophical differences. In addition, there are quite a number of well-known"web hoaxes" where information was published on the Web with the intent to amuse or mislead – the computational agent typically cannot tell the difference! We can summarize much of this even more succinctly by appealing to the punchline of that famous web cartoon: *on the Internet, no one knows you're a dog.*

1.1 Bringing Ontology to the Web

Put succinctly, knowledge representation concerns how a program models what it knows about the world. Years of experience have taught us that a good knowledge representation language is expressive, concise, unambiguous and independent of context. Systems built upon the language should be able to acquire information and perform useful inferences efficiently. The knowledge representation language defines how the information sources are described and the types of queries that are supported. The precision of the representation system limits the precision that is possible in querying it. The inference procedures of the system determine what kinds of queries can be answered.

Many modern representation systems are designed around the useful concept of categorization which allows reasoning about the generality of a concept and allows the careful specification of relationships between these concepts. In recent years, such "ontologies" of concepts have become the focus of much knowledge representation work. In this work, an ontology is a particular theory about being or reality and defines what can exist from a particular perspective.[1] Thus, ontologies allow one to define what is relevant to a particular problem and what should be ignored. Our key contention is that ontologies can be used on the Web to help structure the information – but only if we design the language to take into account many of the web properties described previously. Before we get into the specifics of a knowledge representation language designed for exactly that purpose, we take a moment to reflect on a few of the many web problems that cry out for ontologies as their solutions:

- **Heterogeneity.** Many protocols are used to transmit data on the Internet. These include the Hypertext Transfer Protocol (HTTP), File Transport Protocol (FTP), Telnet, and Gopher. The transmitted data may be in any of a multitude of file formats, including HTML, images, audio, movies, virtual reality modeling files, and others. All of these information sources are potentially useful to someone, but a comprehensive index is necessary to locate anything of value. However, automated indexing is difficult since retrieving semantic information from free-form text and obtaining even minimal information from video or audio sources is practically impossible with current technology. Ontologies can help web users specify, in a formal sense, what information is contained in these information sources, or to search for sources that contain particular types of information.

- **Lack of Structure** The structure of HTML was designed for presentation instead of information retrieval. This structure includes tags for titles, headings, lists and tables, but inconsistent use of these tags makes it difficult to reliably infer semantic meaning from them. As such, it is possible to see if a given word is in a tag, but it is difficult to determine whether a particular concept is the main focus of the page or what the author's viewpoint on a certain subject it is. The only way to locate concepts in HTML is to parse the text in a way that properly interprets synonyms and context. On the other hand, the Extensible Markup Language (XML) [7] will allow authors to create semi-structured documents, but to make full use of XML tags, retrieval mechanisms will have to understand the tag structure used by the document – i.e., they will need some form of ontology. Additionally, XML does not provide any structures for classification or reasoning. However, extending XML-like languages to

[1] As discussed in [24] opinions differ on what is exactly contained in an ontology, however, most agree that it should at least include a taxonomy and describe the valid properties and relations of objects.

include these powerful KR features will allow far more structuring, and the inferential capabilities of KR will allow for knowledge collected from distributed sources to be "fused" via inference rules or other AI mechanisms.

- **Contextual Dependency.** 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 or understanding. Thus, for example an advanced internet search system would need to be able to use context to perform search efficiently. Ontologies provide a mechanism by which context information can be specifically encoded, and a web-based KR language must allow this information to be specified on web pages or in other repositories that refer to web-based information.

To provide these capabilities, we have designed a language named *SHOE*, for Simple HTML Ontology Extensions.[2] SHOE is a KR language that allows ontologies to be designed and used directly on the World Wide Web. In the remainder of this paper we define the language, explaining the syntax, semantics, and how they relate to the issues discussed in this introduction. We then discuss the issue of implementing a complete system the uses SHOE, focusing on the query engine aspect of the system. Then we describe some generic tools (applets and class libraries) that we've developed for SHOE, making it 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 and a comparison to other work in the field. The "old-timer" reading this paper might note that this models the structure of many of the early KR language papers. This is not coincidental – we believe that KR for the Web opens new directions for our field, and that we are still in the early days of this exciting new research.

2 The SHOE Language

SHOE's basic structure consists of *ontologies*, entities which define rules guiding what kinds of assertions may be made and what kinds of inferences may be drawn on ground assertions, and *instances*, entities which make assertions based on those rules. Because SHOE exists in a distributed environment with little central control, SHOE treats assertions as *claims* being made by specific instances instead of facts to gather and intern as generally-recognized truth.

SHOE's syntax is a properly-compliant application extension of HTML; an almost identical XML syntax is also available. However, while SHOE's chief application is the annotation of web documents, SHOE is designed for more general distributed knowledge and distributed agent issues.

In this section, we begin with an overview of the SHOE language, a formal semantic definition of the language, and a discussion of important features of the language. A more complete specification of the language syntax can be found at *http://www.cs.umd.edu/projects/plus/SHOE/spec.html*.

2.1 SHOE Ontologies

In SHOE syntax, an ontology appears minimally between the tags <ONTOLOGY ID=*id* VERSION=*version*> and </ONTOLOGY>. Together, *id* and *version* make up the unique identifier for a particular ontology. Recognizing that knowledge in a distributed environment can change rapidly, SHOE has flexible facilities for ontologies to be derived from one or more superontologies in a multiple-inheritance scheme, or for later versions of ontologies to modify earlier versions; more on this can be found in Section 2.4.6. Figure 1 shows an example of a SHOE ontology.

SHOE defines four basic types: strings, numbers, dates, and boolean values. An additional type, the URL, is under consideration. An ontology may also define additional arbitrary data types on a domain-specific basis. Further, an ontology can make *category definitions* (using the tag <DEF-CATEGORY>) which specify

[2] 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, AltaVista finds 588,508 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.

```
<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 "age">
            <DEF-ARG POS=1 TYPE="Person">
            <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 1: An Ontology Example

```
<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/mike">
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/mike">
        <ARG POS=2 VALUE=me> </RELATION>
    <RELATION NAME="u.age">
        <ARG POS=1 VALUE=me>
        <ARG POS=2 VALUE="32"> </RELATION>
    </INSTANCE>
</BODY>
</HTML>
```

Figure 2: An Instance Example

the categories under which various instances could be classified. For example, an ontology might define the category GraduateStudent. Categories may be grouped as subcategories under one or more supercategories. GraduateStudent might be a subcategory of both Student and Worker, for example.

An ontology can also make *relational definitions* (using structures found within the <DEF-RELATION> and </DEF-RELATION> tags) which specify the format of n-ary relational claims that may be made by instances regarding instances and other data: for example, an ontology might define the relationship advises between an instance of category GraduateStudent and an instance of category Advisor. Or perhaps a relationship age between an instance of category Person and a number (his age).

Lastly, an ontology can define *inferential declarations* within <DEF-INFERENCE> and </DEF-INFERENCE> tags. Inferential declarations specify additional inferences which agents may freely make on ground information. For example, an ontology might specify that working for organizations transfers through to superorganizations, that is, $(\forall x \in \text{Worker})\ (\forall y \in \text{Organization})\ (\forall z \in \text{Organization})\ \text{works-for}(x,y) \wedge \text{suborganization}(y,z) \Rightarrow \text{works-for}(x,z)$.

2.2 SHOE Instances

Instances fill two functions in SHOE. First, instances are arbitrary objects, much like those found in an object-oriented database system. Secondly, instances are the elements in SHOE responsible for making claims.[3] Figure 2 shows an example of a SHOE instance.

In SHOE syntax, an instance appears between the tags <INSTANCE KEY=*key*> and </INSTANCE>. Here, *key* is the instance's unique identifier. For World Wide Web applications, SHOE proposes, but does not formally require, that an instance's unique identifier be based on the URL of the page on which the instance is found; this gives web agents some modicum of ability in determining if an instance really is what

[3] In fact, ontologies are also permitted to make a few specific, highly limited categorization assertions for convenience in ontology design.

it claims to be, by extracting its URL from its key, looking up the web page, and matching its SHOE contents against the instance in question. For the convenience of web site maintenance, instances may specify *delegate instances* who make claims on their behalf. Lastly, ontologies may define simple *constant instances* for basic values like "Red" when defining a custom data type is overkill.

Within an instance may be found ground *category claims* and *relation claims* made by that instance. A category claim is made within the <CATEGORY> tag, and says that the instance claims that some other instance x should be categorized under category y. For example, an instance http://univ.edu/john might claim that http://univ.edu/mike is an Advisor.

A relational claim 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. For example, the instance http://univ.edu/john might claim that there exists the relation advises between http://univ.edu/john and his advisor http://univ.edu/mike. Or http://univ.edu/john might claim that the age of http://univ.edu/mike is 32.

2.3 Formal Definition of the Language

SHOE's semantic knowledge consists of a set of *claims*, made by instances, about relationships between ground atomic elements (numbers, strings, instances, etc.). Claims are either ground claims explicitly stated in instances or claims SHOE has inferred via the simple horn-clause rules defined in an ontology.

2.3.1 Basic Data Sets

SHOE's basic data sets of atomic units are divided into six disjoint sets: BasicTypeName, CategoryName, RelationName, Instance, Ontology, Claim, and Inference. The set Type = CategoryName ∪ BasicTypeName.

Basic Types (members of the set BasicTypeName) have a one-to-one, onto relationship with a special set of objects of that given type. This special set is returned by the function **domain**(x), where $x \in$ BasicTypeName. Each special set has explicit order comparison operators $=$, $>$, $\geq$, $\leq$, $<$ and $\neq$ appropriate for objects of that type. SHOE predefines four data types: String, whose domain is all character strings (comparison is done character-by-character similar to C's `strcmp`); Number, whose domain is all floating-point numbers; Truth, whose domain is {true, false} (false $<$ true); and Date, whose domain is all date/timestamps. This type set may be extended with new types, given proper domains returned by **domain** and an appropriate collection of comparison operator definitions.

The set Instance contains user-declared *instances*, objects which may both make claims of and appear in relationships and categorizations. The set Ontology contains *ontologies*, which may make specific kinds of claims. The set Claimant = Instance ∪ Ontology The set Claim consists of *claims* which claimants may make about the relationships between various data objects. The set Inference consists of *inferential rules* made by ontologies. One special ontology, the *null ontology* κ, is assigned to certain inferential rules described later which are not ordinarily assigned to any other ontology.

2.3.2 Inferential Tags

One peculiarity of SHOE's semantics (its notion of *claim propagation* described below) requires that inferential rules in SHOE be able to refer to themselves as items in their rule bodies. This sets up a potential infinite recursion in the formal definition of inferential rules; we get around this by assigning to each rule a unique *tag* which identifies the rule. This tag may be retrieved with the function **tag**(i), which returns the tag for a given inference i. When rules need to refer to themselves in their rule bodies, they instead use these tags.

2.3.3 Claim Propagation

SHOE does not consider statements it discovers on the web as *facts*, but as *claims* made by a particular claimant. When performing inferences on these claims, SHOE must propagate the claimants from the antecedents of the inference into the consequents of the inference (that is, if claimant A said a, and claimant B said b, and a SHOE inference exists which says $a \wedge b \Rightarrow c$, SHOE must determine who "said" c).

SHOE does this by propagating user-defined objects called *claim groups*. SHOE requires two user-defined functions which manipulate claim groups: $\mathbf{claim}(j, I)$ takes a label j for a claim $C \in \mathsf{Claim}$ made by a set of claimants $I \in \mathsf{Claimant}$, and returns a claim group q. $\mathbf{prop}(Q, j, O)$ takes a set of claim groups Q generated by subclaims to a given inference whose tag is j, plus a set of ontologies $O \subseteq \mathsf{Ontology}$ which defined the inference, and returns a new claim group q.

SHOE does not define these transfer functions nor the implementation of claim groups, and their use is an active area of research. We feel that various SHOE agents may need different amounts of claim information, depending on their domain. A sophisticated agent might use these functions to gather enough information to trace inferred claims back through the entire tree of inferences and ground claims to determine exactly how an inferred claim was determined. However, this is an expensive process. An unsophisticated agent might not care about who claimed what, and define the functions appropriately. We think a reasonable, inexpensive default version of these functions is $\mathbf{claim}(h, I)$ to simply return I, and for $\mathbf{prop}(Q, j, O)$ to return $O \cup (\bigcup Q)$ This version would return a set of claimants jointly making a claim, without detailing how they each came to be part of the joint claim.

2.3.4 Category Definitions

Tuples of the form $\langle c, o, \{i : i \in \mathsf{CategoryName}\}\rangle$, are *category definitions*. The first element in a well-formed category definition, $c \in \mathsf{CategoryName}$, is the definition's *category name*. Category definitions have unique category names: there is a one-to-one, onto relationship between each symbol c and a category definition $\langle c, o, \{i : i \in \mathsf{CategoryName}\}\rangle$. We define a function $\mathbf{cat}(c)$ which takes a symbol $c \in \mathsf{CategoryName}$ and returns its corresponding category definition tuple.

The second element in a well-formed category definition is an ontology $o \in \mathsf{Ontology}$ which made this definition. o may not be κ.

The third element in a well-formed category definition is a (possibly empty) set of category names of which c is a *subcategory*. A category is automatically its own subcategory. Hence we define a predicate relation $\mathbf{subcat}(c, b)$ which is true if and only if c is a subcategory of b. Formally, $\mathbf{subcat}(c, b) \Leftrightarrow (\exists o)(\exists I)(\mathbf{cat}(c) = \langle c, o, I\rangle) \wedge (b \in I \vee b = c)$.

Each subcategorization entry in a category definition automatically adds an inferential rule to the set Inference which defines that subcategorization. For a given category definition $\langle c, o, \{..., b, ...\}\rangle$ the inferential rule (with some unique tag j) describing the subcategory relationship between c and b takes the form:

$$(\forall d \in \mathsf{Instance})(\forall q)\mathrm{cclaim}(c, d, q) \wedge \mathbf{subcat}(c, b) \Rightarrow \mathrm{cclaim}(b, d, \mathbf{prop}(\{q\}, j, o))$$

The predicate cclaim(...), which describes a categorization claim, is described below in Section 2.3.6. The inference roughly says that for each categorization claim on a given instance for a given category, there is an implicit categorization claim made on the instance for every supercategory of that category.

2.3.5 Relation Definitions

Tuples of the form $\langle r, \langle x_0, ...\rangle\rangle$ are *relation definitions*. The first element in a well-formed relation definition, $r \in \mathsf{RelationName}$, is the definition's *relation name*. Relation definitions have unique relation names: there is a one-to-one, onto relationship between each symbol r and a relation definition $\langle r, \langle x_0, ...\rangle\rangle$. We define a function $\mathbf{rel}(r)$ which takes a symbol $r \in \mathsf{Relation}$ and returns its corresponding relation definition tuple.

The second element in a well-formed relation definition is a nonempty tuple of arguments. Each argument x_n in this tuple must be a member of Type.

2.3.6 Categorization Claims

Categorization claims, made from a set of *claimants* I, state that the claimants claim that instance d belongs to the category c. A claim C with label j of the form $\mathrm{cclaim}(c, d, \mathbf{claim}(j, I))$ may be added to the set Claim only if:

1. $c \in \mathsf{Category}$
2. $d \in \mathsf{Instance}$

3. $I \subseteq$ Claimant.

2.3.7 Relation Claims

Relation claims, made from a set of claimants I, state that the claimants claim that a relation r exists between the ordered arguments $\langle d_0, ...\rangle$. A claim R with label j of the form $\text{rclaim}(r, \langle d_0, ...\rangle, \mathbf{claim}(j, I))$ may be added to the set Claim only if:

1. $r \in$ Relation
2. $I \subseteq$ Claimant.
3. The claim properly matches, type-for-type, the relation definition for r. That is, there exists a $\langle r, \langle x_0, ...\rangle\rangle = \mathbf{rel}(r)$ such that for each $\langle ..., d_n, ...\rangle$ and corresponding $\langle ..., x_n, ...\rangle$,
   (a) if $x_n \in$ CategoryName, then $d_n \in$ Instance
   (b) if $x_n \in$ BasicTypeName, then $d_n \in \mathbf{domain}(x_n)$

Relation claims occur under a special inferential rule, an explicit member of the set Inference. Let the tag of this rule be j. Then the rule is:

$$(\forall n \in \{0, ..., ||\langle d_0, ...\rangle|| \Leftrightarrow 1\})(\forall a),$$
$$\text{rclaim}(r, \langle ..., d_n, ...\rangle, q) \wedge \mathbf{rel}(r) = \langle r, \langle ..., x_n, ...\rangle\rangle \wedge x_n \in \text{CategoryName} \Rightarrow \text{cclaim}(x_n, d_n, \mathbf{prop}(\{q\}, j, \kappa))$$

Simply put, this rule says for each relation claim, if an argument of that relation has been typed as a category, then there is another implicit claim that the object occupying that position in the relation claim is an instance of the specified category. 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. Because SHOE uses an open-world policy, there is no way to know that a given object in a relation claim is *not* in a category appropriate for that relation; since the categorization claim may as yet be undiscovered.

2.3.8 Inferential Declarations

To the inferential rules described above, ontologies may add to the set Inference additional inferential rules with certain constraints, called *inferential declarations*. If these declarations are well-formed, SHOE will intern them as additional inferential rules to apply at query-time.

A SHOE inferential declaration has an *antecedent* and a *consequent*,[4] separated with a $\Rightarrow$ ("implies") operator (the consequent is on the right). At certain places in an inferential declaration, constants may be replaced by *claim variables*, which are nothing more than the variables in the logical inference being made. All instances of a particular claim variable appearing in a well-formed inferential declaration are associated with a given member of the set Type; this member is returned by the function $\mathbf{vtype}(v)$, where v is the claim variable in question. Further, such variables adhere to a *join rule* described later.

The consequent of a well-formed SHOE inferential declaration is a single well-formed relation or category subclause, prepended with one universal quantifier $(\forall v)$ for each variable v appearing anywhere in the inferential declaration. The antecedent of a well-formed SHOE inferential declaration is a nonempty conjunction of well-formed relation subclauses, category subclauses, and comparison subclauses. Let m be the total number of relation and category subclauses in the antecedent. Further, let j be the tag of the inferential declaration, and $O \subseteq$ Ontology be the set of ontologies which make this inferential declaration.

[4] In logic programming these are often referred to as the body and head, respectively.

Relational subclauses. A relation subclause $\text{rclaim}(r, \langle d_0, ... \rangle, q)$ is well-formed if:

1. $r \in \mathsf{Relation}$
2. If the relation subclause is in the antecedent, then q is written as q_n, where n is the position of the subclause in a left-to-right ordering of relation and category subclauses in the antecedents. If the relation subclause is in the consequent, then q takes the form $\mathbf{prop}(\{q_0 ... q_{m-1}\}, j, O)$.
3. The claim properly matches, type-for-type, the relation definition for r. That is, there exists a $\langle r, \langle x_0, ... \rangle \rangle = \mathbf{rel}(r)$ such that for each $\langle ..., d_n, ... \rangle$ and corresponding $\langle ..., x_n, ... \rangle$,
   (a) d_n is either a constant or a claim variable.
   (b) if $x_n \in \mathsf{CategoryName}$, then $d_n \in \mathsf{Instance}$ or, if d_n is a variable, $\mathbf{subcat}(\mathbf{vtype}(d_n), x_n)$
   (c) if $x_n \in \mathsf{BasicTypeName}$, then $d_n \in \mathbf{domain}(x_n)$ or, if d_n is a variable, $\mathbf{vtype}(d_n) = x_n$

Category subclauses. A category subclause $\text{cclaim}(c, d, q)$ is well-formed if:

1. $c \in \mathsf{Category}$
2. If the relation subclause is in the antecedent, then q is written as q_n, where n is the position of the subclause in a left-to-right ordering of relation and category subclauses in the antecedents. If the relation subclause is in the consequent, then q takes the form $\mathbf{prop}(\{q_0 ... q_{m-1}\}, j, O)$.
3. d is either a constant or a claim variable.
4. $d \in \mathsf{Instance}$ or, if d is a variable, $\mathbf{subcat}(\mathbf{vtype}(d), c)$

Comparison subclauses. A comparison subclause $d_0\ e\ d_1$ is well-formed if:

1. d_0 is a claim variable.
2. d_1 is is either a constant or a claim variable.
3. if d_1 is a claim variable, then $\mathbf{vtype}(d_0) = \mathbf{vtype}(d_1)$.
4. if d_1 is a constant and $\mathbf{vtype}(d_0) \in \mathsf{BasicTypeName}$, then $d_1 \in \mathbf{domain}(\mathbf{vtype}(d_0))$.
5. if d_1 is a constant and $\mathbf{vtype}(d_0) \in \mathsf{CategoryName}$ then $d_1 \in \mathsf{Instance}$.
6. if $\mathbf{vtype}(d_0) \in \mathsf{BasicTypeName}$, then e is one of $=, \neq, \leq, <, \geq$, or $>$, else if $\mathbf{vtype}(d_0) \in \mathsf{CategoryName}$, then e is one of $=$ or $\neq$.

The variable join rule. Variables in well-formed inferential subclauses adhere to the following join rule. For all variables x and y in the antecedents of an inferential subclause, the predicate function $\mathbf{join}(x, y)$ is defined as:

1. $\mathbf{join}(x, y)$ is true if x and y appear in the same relation subclause, or in the same comparison subclause only when $\mathbf{vtype}(x) = \mathbf{vtype}(y) \in \mathsf{CategoryName}$ and e is set to $=$.
2. For any variable z in the antecedents, $\mathbf{join}(x, y) \wedge \mathbf{join}(y, z) \Rightarrow \mathbf{join}(x, z)$.

Then an inferential declaration is well-formed if and only if:

1. For all x and y variables (where $x \neq y$) in the antecedents of the declaration, $\mathbf{join}(x, y)$ is true.
2. For each variable z appearing in the consequent, z also appears in the antecedent.

Note that this rule has more severe constraints than the "safe" rule defined for Datalog. In Datalog, variables do not need to be fully joined as they are in SHOE; hence Datalog permits cartesian products over relations directly in its horn clauses. However, SHOE can implement these features using multiple inferential declarations. Thus there is no loss of semantic expressivity while accidental or careless cartesian products in ontologies, which can be very expensive over large amounts of data, are avoided.

2.4 Language Features

SHOE was designed specifically with the needs of distributed internet agents in mind. In this section we discuss the features that make SHOE well-suited for this purpose.

2.4.1 Compatibility with HTML/XML.

Even though the Web is heterogeneous at the content level, there are still standards at the protocol level and file format level. Since HTML is the primary format for Web documents, SHOE was designed to fit seamlessly into HTML. We use a similar syntax that makes it easy for those who know HTML to learn SHOE. SHOE is an application of the Standard Generalized Markup Language(SGML) [15]: its HTML-compatible syntax is formally defined in an SGML document type declaration (DTD) that is derived from the formal HTML DTD.

A slight variant of the SHOE syntax also exists for compatibility with XML [7], the emerging standard for transmitting web documents. XML is in effect a simplified form of SGML, and thus the XML syntax for SHOE is almost identical to the original syntax. When XML becomes commonplace, the XML variant of SHOE is likely to become the standard. Like SGML, XML is a language for defining and using markup languages. Thus it is possible to define a tag set that relates to content rather than presentation, but there is no way to define the meaning of this content in a machine-understandable way. Additionally, interoperability between domains must be hard-coded into applications. It is for these reasons that a language like SHOE is needed.

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), thus software that is not SHOE-aware may still use the information in more limited but still 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 emerging XML standard for stylesheets to render SHOE information for human consumption. This is perhaps the most important reason because it eliminates the redundancy of having a separate set of tags for the human-readable and machine-readable knowledge.

2.4.2 Prevention of Contradictions.

As there is no easy way to control what distributed agents may say, SHOE's design philosophy avoids 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 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 includes the claimant as part of a claimed assertion.

SHOE does not prevent "contradictions" that are not logically inconsistent. If claimant A says father(Mark, Katherine) and claimant B says father(Katherine, Mark), the apparent contradiction is because one claimant is misusing the father relation. However, this does not change the fact that A and B made those claims.

Similar problems may also 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.

2.4.3 Separation of Ontologies and Instances.

Unlike other distributed knowledge systems, (notably the Resource Description Framework (RDF) [18]), SHOE intentionally separates ontologies and instances into separate syntactic objects. Our methodological justification is simple: agents should have the freedom to intern information from all instances, while placing restrictions on which ontologies they intern. It's bad enough that entity A makes a false claim; it's far worse when entity A adds whole false inferences into an agent's internal ontology. By separating the rules guiding claims from the claimants making the claims, SHOE gives agents some flexibility in this regard.

Additionally, ontologies are used to validate and interpret instances. The ontology defines the vocabulary that an instance may use, and could be compared to a schema in a relational database or a DTD in SGML or XML. However, the ontology also includes information that serves as a machine-readable definition of the terms, and allows systems to reason about claims that use terms from the vocabulary.

2.4.4 N-ary Relations.

Unlike several distributed knowledge systems, SHOE permits n-ary relations. Although in theory binary relation semantics suffices, we feel that in practice it hobbles the ontology designer and claimants in unnecessary ways.

2.4.5 Uniqueness of Identification.

A particular problem with natural language, especially on the Web, is that it is ambiguous. A system processing information must cope with the problems of *synonymy* (when multiple words have the same meaning) and *polysemy* (when a single word has multiple meanings). In SHOE, these problems are avoided because both ontological elements and instances are uniquely identified.

Each ontology element has exactly one definition and an ontology may not contain elements with the same name. Although different ontologies may use the same name to define a different concept, ontological elements are always referenced using special prefixes which define unique paths leading 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. Furthermore, an ontology can specify an alternate label for another ontological element, thus creating a synonym that can still be resolved to the object in which the concept was originally defined.

In the case of instances, each must be assigned a unique 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. However, many different URLs could be used to refer to the same page, since the host can have multiple domain names and operating systems may allow many different paths to the same file. To solve these problems a canonical form must 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.

2.4.6 Extensibility and Versioning.

SHOE attempts to achieve information integration through the use of shared ontologies. When two ontologies need to refer to a common concept, they should both extend an ontology in which that concept is defined. Extension also allows an ontology to be customized by a particular community, so that it can include definitions and rules for specialized areas of knowledge.

Of course, in a distributed environment ontologies may need to change rapidly. The challenge is in providing flexible ontology modification while maintaining unification across various ontologies and without invalidating SHOE that references old ontology versions. To accomplish these objectives, each version of a SHOE ontology is a separate file and is assigned a unique version number. In this way all versions of an ontology are accessible at any point in time. Further, an ontology can specify that it is backwardly-compatible

with earlier versions. This allows computer systems to use the new ontology to correctly interpret SHOE annotations that reference older versions.

It should be noted that the versioning mechanism is dependent on the compliance of the ontology designers. There is nothing to prevent a designer 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.

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.

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 [14].

3 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 design issues of 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.

3.1 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. To implement all of SHOE, a system must be able to handle at least the following features:

- support for n-ary predicates
- inference of category membership
- constrained horn-clause inference without negation, procedural attachment, or cartesian products
- built-in comparison predicates such as $=$, $<$, and $>$

Depending on domain requirements, the back-end engine for a SHOE agent may also need to implement claim maintenance and propagation to some degree or other; at one extreme, it is perfectly reasonable for engine to ignore claims entirely. 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. But SHOE's goal is to balance semantic expressivity with the computational complexity *in practice* given reasonable assumptions about the nature of a distributed domain like the World Wide Web. In the

following section we discuss various design decisions in SHOE and how to exploit these to implement SHOE efficiently. Afterward, we discuss implementing SHOE, or parts of SHOE, in three semantically efficient systems: Parka [11, 24], a high-performance KR system whose roots lie in semantic networks and object-based systems, Datalog [25], a popular forward-chaining logic system, and lastly an ordinary relational database management system.

3.1.1 Approaches to Efficient SHOE

The two general areas of efficiency concern in SHOE are its inheritance mechanism, and its general inferential rules. Efficient inheritance has traditionally been problematic for more expressive semantic network systems, but this is usually because of default logics, prototypes, inferential distance ordering, and other inheritance features which SHOE does not support. Even so, in Parka we have demonstrated highly efficient algorithms for handling inheritance more semantically expressive than SHOE's. These algorithms scale exceptionally well, handling millions of assertions with relative ease. [24]

As in Datalog, the implementation of SHOE's general inferential rules can be done naively in polynomial time and space in the worst case, which is pretty good. But further, SHOE takes advantage of an important heuristic: the thesis 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.

Because SHOE has monotonic, limited inferential semantics without negation, it is straightforward to take advantage of this heuristic to determine at query-time exactly those rules relevant to the query in $O(n)$ time, where n is the number of relevant rules. With a highly localized rulebase, this can result in dramatic savings. The procedure is trivially done as follows: Let G be a set of subgoals to achieve, initially set to the subgoals in the requested query. Let R be the set of rules necessary to achieve the query, initially set to $\emptyset$. Remove a subgoal g from G; for each rule r not in R whose consequent achieves g, add r to R and add to G each subgoal in the antecedent of r. Repeat until G is exhausted.

These same SHOE semantics also allow agents to determine whether or not the relevant rules described above are recursive, which helps an agent efficiently pick between backward- and forward-chaining, or to pick more efficient forward-chaining algorithms. This can be done in $O(n^2)$ time worst-case, where n is again the size of the set of relevant rules discussed before. This procedure is also simple: Let R be the set of rules determined above. For each $r \in R$, set $s_r \in S$ to the set of rules in R which achieve a subgoal in the antecedent of r. With hashing this is $O(n^2)$ at worst. This then reduces to a problem of finding a cycle in a graph with R as vertexes and S as edges, which is $O(n)$.

3.1.2 Parka

Parka is a high-performance knowledge base system developed at the University of Maryland, which performs complex queries over very large knowledge bases in less than a second. At present, Parka is the base KR system on which we have built our SHOE back-end. 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 has no direct support for claims: though it might represent claimant information with an extra argument to each predicate. However, the structural links (i.e., the links along which inheritance is performed) are built-in binary predicates in Parka. Thus, this approach could not be used for isa or instanceof predicates without changing the internal workings of the knowledge base. To keep the storage of claim information consistent for relations and categories, we can instead make two assertions for each claim. The first assertion ignores the claimant, and can be used normally in Parka. The second assertion uses a claims predicate to link

the source to the first assertion. When the source of information is important, it can be retrieved through the claims predicate. Although this results in twice as many assertions being made to the knowledge base, it preserves inheritance while keeping queries straightforward.

Since Parka has no general inferential capabilities, if SHOE inference rules are important to an agent, their implementation would need to be outside knowledge base system. An inference engine could be written that interfaces with the Parka KB, although this would obviously be slower than in a system in which inference was built in.

3.1.3 Datalog

Datalog is a forward-chaining deduction system for databases based on first-order predicate logic using horn clauses. Datalog has good computational complexity, however it is in general less efficient than a custom-built SHOE system, especially with regard to category inheritance.

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 a predicate, 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 has a good match with SHOE's semantics: it allows n-ary predicates, horn-clause rules, and built-in comparison predicates. SHOE's more restrictive variable join rule ensures that all SHOE inference rules are safe. And since they have no negation, SHOE inferential rules map directly to recursive Datalog horn clause rules. The remaining semantics of SHOE can be achieved using Datalog's rules. Category membership can be inferred using Horn clause rules and unary predicates. For example, the fact that a Person is a Mammal may be expressed using the rule Person(x) $\Rightarrow$ Mammal(x).

Datalog, as well as most other knowledge base systems, does not provide a mechanism for attaching sources to assertions or facilities for treating these assertions as claims. To represent such information, one must create an extra layer of structure using the existing representation. To represent the source of the information, for example, we can add an extra argument to each predicate indicating the source. However, claim propagation can only be done in simplistic ways through modifications of SHOE rules as they are converted through Datalog rules.

3.1.4 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. To accommodate claimant information, an additional attribute can be added to each relation as in the two systems above.

The only thing missing from RDBMSs is inference. However, 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. Ullman describes this process in detail in [25]. As mentioned earlier, the semantics of SHOE are safe and include no negation, but SHOE rules can be recursive. Therefore, some but not all of the rules could be implemented using views. 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.

3.2 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, information gathering, and information processing.

3.2.1 Ontology Design

Ontology design can be a time consuming process. To save time and increase interoperability with other domains, an ontology designer should determine what set of existing ontologies 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 [26].

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.

3.2.2 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 3.3.2.

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 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.

3.2.3 Information Gathering

A system that uses SHOE information can be reactive or proactive in terms of gathering information. A reactive system will only scan a page that has been explicitly specified by some user, perhaps via an HTML form. A proactive system, on the other hand, seeks out new web pages in an attempt to locate useful information. 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. The first two cases are considered *off-line* processing because information is stored locally and queries are issued against this local KB instead of the web at large; this is similar to the way major search engines work today. The other case is considered *on-line* processing because query execution involves dynamically loading web pages in a very specific search. One advantage of the off-line approach is that accessing a local KB is much faster than loading web pages. A second advantage is that we do not need complex heuristics to achieve a competent gathering of the information. Instead, the system stores all information that it comes across for later user. 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.

3.2.4 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 3.1, 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.

3.3 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.

3.3.1 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.

3.3.2 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 3, 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,

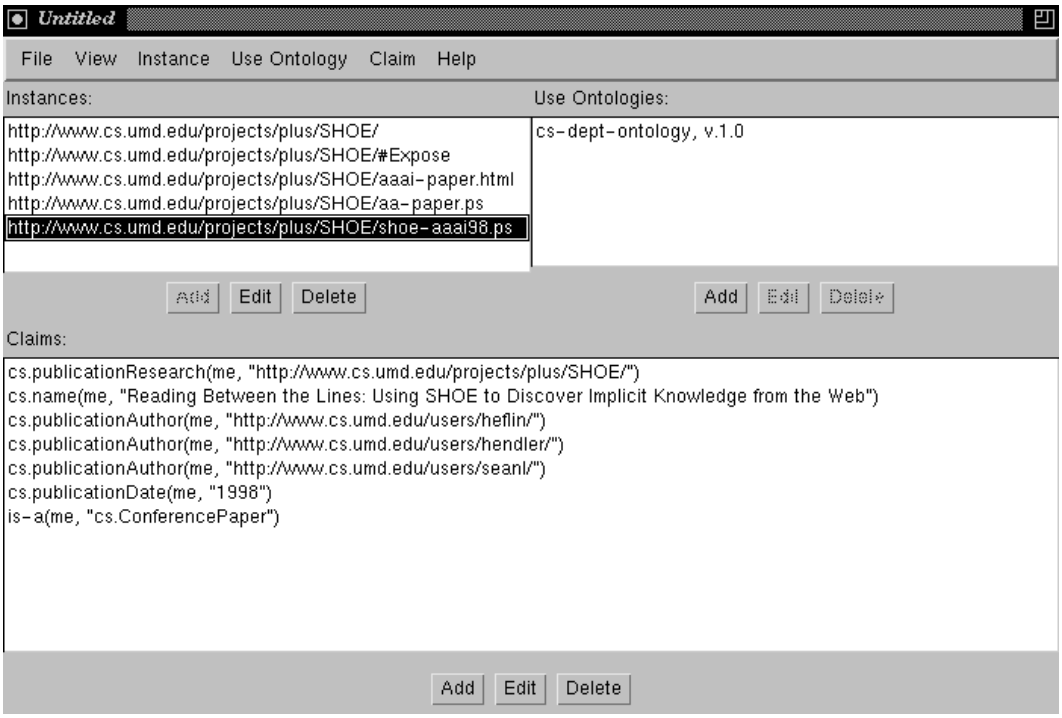

Figure 3: The Knowledge Annotator

users are prompted for the necessary information. In the case of claims, a user can choose the source ontology from a list, and then choose categories or relations from a corresponding list. The available relations will automatically filter based upon whether the instances entered can fill the argument positions. A variety of methods can be used to view the knowledge in the document. These include a view of the source HTML, a logical notation view, and a view that organizes claims by subject and describes them using simple English. In addition to prompting the user for inputs, the tool performs error checking to ensure correctness[5] and converts the inputs into legal SHOE syntax. For these reasons, only a rudimentary understanding of SHOE is necessary to markup web pages.

3.3.3 Exposé

Exposé is a web-crawler that searches for web pages with SHOE mark-up and interns the knowledge. It can be used to gather information for an off-line search system. Exposé can be configured to visit only certain web sites or directories within those web-sites. This allows the search to be constrained to 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 also follows web robot etiquette: it will not request any page that has been disallowed by a server's robot.txt file and waits 30 seconds between page requests, so as not to overload a server.

A web-crawler essentially performs a graph traversal where the nodes are web pages and the arcs are the hypertext links between them. When Exposé discovers a new URL, it assigns it a cost and uses this cost to determine where it will be placed in a queue of URLs to be visited. In this way, the cost function determines the order of the traversal. We assume that SHOE pages will tend to be localized and interconnected. For this reason, we currently use a cost function which increases with distance from the start node, where paths through non-SHOE pages are more expensive than those through SHOE pages and paths that stay within the same directory on the same server are cheaper than those that do not.

[5] 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.

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 stores SHOE category and relation claims in a specified knowledge base; a small set of interface functions must be written to provide an API for the KB.

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 the source has made an implicit category claim: we assert a claim that the instance is of the required category.

4 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 to solve a real-world problem. The design of these applications followed four basic steps:

- create the ontologies
- annotate web pages with SHOE tags
- determine how SHOE information is to be discovered and processed
- design the user interfaces

4.1 A Computer Science Department Application

Our 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 small-size environment. Our basic architecture consists of using the Knowledge Annotator to add SHOE tags to web page, using Exposé to discover knowledge, and using a graphical Java applet to query the knowledge base that stores the knowledge.

4.1.1 The Approach

We began by creating a simple computer science department ontology that extends the base SHOE ontology. Some of the categories we defined were Student, Faculty, Course, Department, Publication and Research. We also defined relations such as publicationAuthor, emailAddress, advisor, and member. The final ontology had 43 categories and 25 relations. We created the ontology file[6] by hand, and included 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. Annotation is the process of adding SHOE semantic markup to a web page. 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.

In order to get a better understanding of the system's performance, we needed a larger dataset than just the pages of the PLUS group. Fortunately, the World Wide Knowledge Base (WebKB) project at Carnegie Mellon University has a dataset in the same domain[7]. This set consists of 8282 real web pages found at computer science department websites and includes seven manually generated classifications for the web pages. These classifications include four categories (Course, Faculty, Student, and Research) and three binary relations (suborganization, teacherOf, and member). We used this information to create category and relation declarations in SHOE format. Since many of the web pages did not fall into any of the categories used, this only allowed us to acquire 1929 new claims. Since, for obvious reasons, we were unable to add this SHOE

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

[7] This dataset can be obtained from *http://www.cs.cmu.edu/~webkb/*

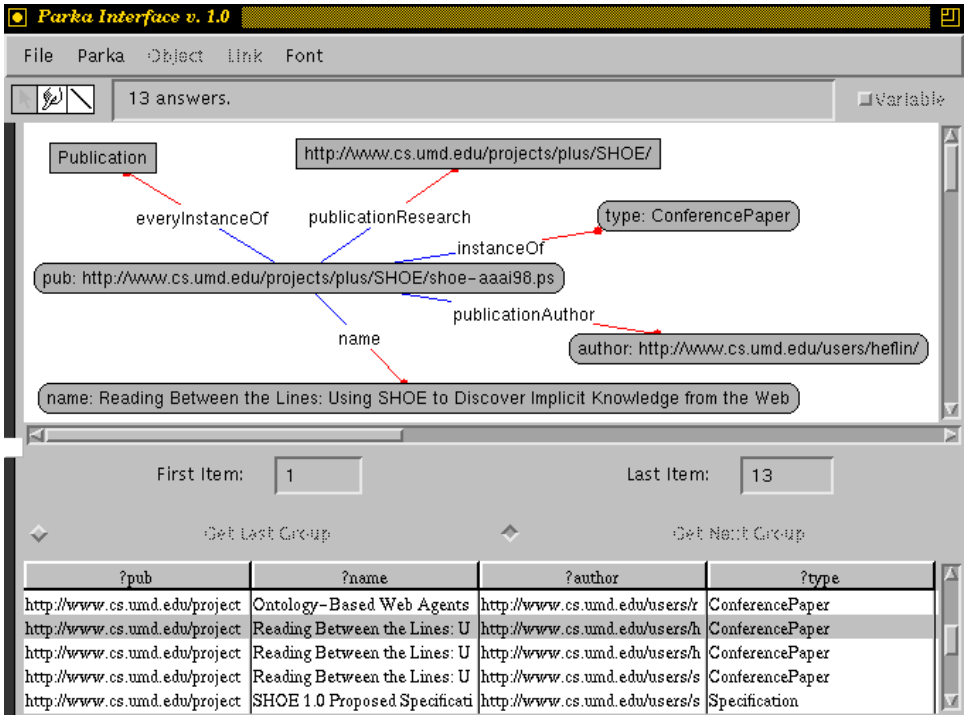

Figure 4: A Java-PIQ Query

data to the web pages of the departments described, we created summary pages on our server to contain the resulting SHOE instances.

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 then 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 decided to use Exposé to acquire the SHOE knowledge from the web pages. To store the results of Exposé's search, we chose Parka, since we were familiar with it and could readily modify it if needed. As mentioned in Section 3.1.2, Parka possesses most of the features required for a system to implement the full SHOE semantics; it only lacks the ability to do arbitrary inference. Since we were not interested in performing complex inferences on the data at the time, this was of no consequence. Not including the intensional delay between page requests, it took the robot 332 seconds to scan the directories and load the SHOE data into the Parka KB.

Once we had the data loaded in SHOE, we needed to be able to query the Parka KB to make use of it. The Java Parka Interface for Queries (PIQ) was developed for just this purpose. This interface gives users a new way to browse the web by allowing them to submit complex queries and open documents by clicking on the URLs in the results. Figure 4 displays the results of a query to find the names, authors, and classifications of every paper published on SHOE. A user inputs a query by drawing frames and the relations between them. This specifies a conjunctive query in which the frames are either constants or variables and the relations can be a string matching function, a numerical comparison or a relation defined in an ontology. The PIQ is an applet, and as such is executed on the machine of each user who opens it. This client application communicates with a central Parka knowledge base through a Parka server that is located on the website that makes the PIQ available. When a user starts a PIQ applet on their machine, this applet sends a message to the Parka server. The server responds by creating a new process and establishing a socket for communication with the applet. Then the server processes the query and sends the answers back to the

applet, where they are displayed as a table of the possible variable bindings. 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.

4.1.2 Discussion

As this example demonstrates, there are many possible means of acquiring SHOE information. In our prototype, we used human directed annotation, translation from an alternate format, and pattern extraction. This allowed us to develop a significant number of SHOE assertions in days. By combining the ability to annotate one's own pages with the ability to make claims about the content of other web pages, we allow the efforts of information providers and professional indexers to be combined.

The most difficult task for the individual annotating is 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 claims 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.

Although our KB is very small when compared to the scale of the entire Web, the initial results are promising. For example, a Java-PIQ query of the form member(http://www.cs.umd.edu, x) $\wedge$ instance(Faculty, x) took only 256 milliseconds to answer, and the majority of this time was spent communicating with the server across the internet. Future work will involve a thorough performance evaluation using much larger knowledge bases and arbitrary inference.

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. One of the biggest advantages of SHOE is that the data contained in multiple sources can be combined to answered a single query.

4.2 The "Mad Cow Disease" Application

The "Mad Cow Disease" epidemic in Great Britain and the apparent link to Creutzfeldt-Jakob disease (CJD) in humans generated an international interest in these diseases. Bovine Spongiform Encephalopathy (BSE), the technical name for "Mad Cow" Disease, and CJD are both Transmissible Spongiform Encephalopathies (TSEs), brain diseases that cause sponge-like abnormalities in brain cells. Concern about the risks of BSEs to humans continues to spawn a number website on the topic.

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 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 with using SHOE to support the TSE Risk Website, the first step in building a food safety clearinghouse.

4.2.1 The Approach

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

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

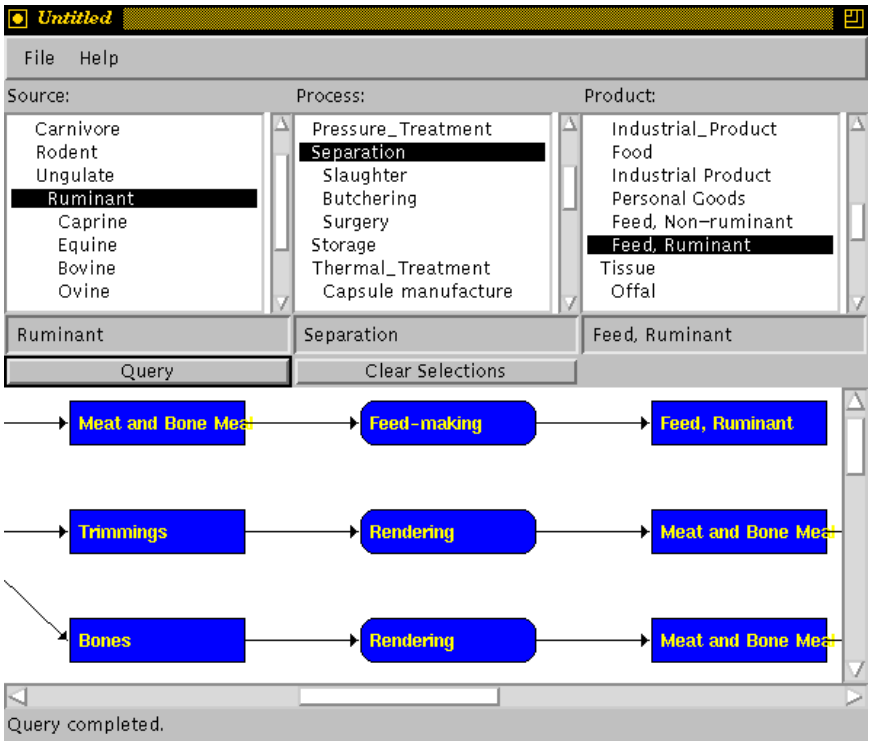

Figure 5: The TSE Path Analyzer

addition to specific TSE concepts such as Disease and Risk, general terms such as People, Organizations, Processes, Events, and Locations were defined. Twelve of the relations have three or more arguments, confirming our intuitions of 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, we 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 and used Parka as our knowledge base system. The resulting KB could be queried using the PIQ, 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 which 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 5. 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 almost the same as described above. A summary of the process is below:

1. Knowledge providers who wish to make material available to the TSE Risk Website use the Knowledge Annotator to mark-up their pages with SHOE. 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.

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 the PIQ.

It is important to note that we will force new websites with TSE information 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.

4.2.2 Discussion

The approach taken in developing this application was similar to that used for the computer science department application. However, there were two factors that made this more difficult. First, the domain had ill-defined boundaries forcing us to resort to an iterative process of ontology design and page annotation. Second, the annotators and the query users were not familiar with AI techniques and terminology.

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. In the course of developing the TSE ontology, we developed the following guidelines to help us determine 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?

The annotation process was more difficult for this application, partly because the annotators were either AI researchers or domain experts, but not both. However, we believe the main difficulty was due to the difference in the data. In the computer science application, instances of each major concept (e.g., department, course, professor, publication) has its own home page. The content of TSE pages on the other hand mostly refers to shared entities such as BSE or the North American continent, and thus choosing a single URL as a key is difficult. In such cases, we created constants in the ontology to represent the shared objects, although we are investigating better alternatives.

Users also have difficulty in determining the level of detail that is required when annotating a particular page. The more detailed the information, the more likely the page is to be returned to a precise query that matches its contents. However, adding detailed 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 claims.

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 the Java PIQ, even though the former only allows a much more limited set of possible queries. However, users liked it because it required little instruction and displayed the results in a customized way that made it easy to explore the problems of interest to them.

We also learned that a SHOE knowledge base must be able to perform certain complex operations as a single unit. For example, an important feature of the TSE Path Analyzer is that it displays a hierarchical lists of sources, processes, and products where each list is built from the categorization hierarchy. Although theoretically these lists can be built recursively by asking for the immediate children of a category, in client-server situations the communication overhead and transmissions delays for each request can be prohibitive. Much better performance is achieved with a special server request that returns the complete set of parent-child pairs that form a hierarchy. Although this required the same amount of processing by the knowledge base, it resulted in a significant speedup of the client application.

5 Related Work

Attempts to bring semantic markup to the Web are not new. In fact, a limited form of semantic markup is present in early versions of HTML. HTML 2.0 [3] includes several weak mechanisms for semantic markup (the REL, REV and CLASS subtags and the META tag). HTML 3.0 [22] advances these mechanisms somewhat. Unfortunately, the semantic markup elements of HTML have so far been used primarily for document meta-information (such as declared keywords) or for hypertext-oriented relationships (like "abstract" or "table of contents"). Furthermore, relationships can only be established along hypertext links (using <LINK> or <A>).

To address the limitations of HTML, Dobson and Burrill [9] have attempted to reconcile it with the Entity-Relationship (ER) database model. This is done by adding to HTML 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. However, no inferencing mechanism is provided.

Many research projects aim to make the Web more productive by adding database functionality. Some projects focus on creating query languages for the Web [1, 17]. However 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. Another approach involves mediators (or wrappers), custom software that serves as an interface between middleware and a data source [27, 21, 23]. 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. Other researchers [8] are looking at using machine learning techniques to categorize web pages and extract relationships from the text.

The concepts of creating ontologies by extending pre-existing ontologies has been used in numerous KR systems. One of the most notable of these is the Ontolingua Server [12] which is a tool to assist in the design and sharing of ontologies. The idea of partitioning KBs using ontologies is very similar to Cyc's [19] microtheories.

The projects most similar to SHOE focus on creating languages to help machines process and understand Web documents. The Ontobroker [13] project has developed a language which is embedded in HTML as well. The syntax of this language is more compact but is not as easy to understand as SHOE. Also, Ontobroker does not have a mechanism for pages to use multiple ontologies and those who are not members of the community have no way of discovering the ontology information. The Web Analysis and Visualization Environment (WAVE) project [16] has designed the Ontology Markup Language (OML) and the Conceptual Knowledge Markup Language (CKML), both of which are based on early versions of SHOE.

5.1 The Resource Description Framework

The Resource Description Framework (RDF) [18] is a recommendation endorsed by the World Wide Web Consortium (W3C). Since RDF is similar in concept to SHOE but has the backing of the major Internet standards body, we feel it is necessary to provide a detailed comparison of SHOE and RDF. First, RDF is not a language, but a data model of metadata instances. Additionally, this data model is nothing more than a semantic network without inheritance; it consists of nodes connected by named links. To include RDF in files, its designers have chosen an XML syntax although they emphasize this is only one of many possible representations of the RDF model. This syntax is used to describe the properties and relations of resources in much the same way as relations are used in SHOE. A property called rdfs:type is used to express the type of a resource; this is equivalent to the <CATEGORY> tag in SHOE. The most significant difference between the RDF syntax and SHOE instances is that being based on a semantic network, RDF is inherently binary. However, a version of the syntax makes this fact almost transparent to users.

Following SHOE, RDF has recognized the need for controlled, sharable, extensible vocabularies if interoperability is to be achieved on the Web. To this end, the RDF working group has developed the RDF Schema Specification. We can compare RDF schemas to SHOE ontologies. RDF defines a property called rdfs:subClassOf which is equivalent in semantics to SHOE's <DEF-CATEGORY>, with the exception that if there are multiple parent categories, each must be specified by repeating the property. One feature possessed by RDF that does not appear in SHOE is the property rdfs:subPropertyOf which allow properties to be specialized in a way similar to classes. However, the same semantics can be expressed in SHOE using inference rules. RDF allows constraints to be placed on the domain and range of a property; this is equivalent to specifying the type of arguments in SHOE <DEF-RELATION> statements. RDF does not possess any mechanisms for defining general inferences.

In RDF, schemas are extended by simply referring to objects from that schema as resources in a new schema. Since schemas are given unique URIs (Uniform Resource Identifiers), this guarantees that exactly one object is being referenced. This achieves the same purpose as <USE-ONTOLOGY> in SHOE, but in a much less compact way. A significant oversight in RDF is the lack of an ability for a schema to rename properties and classes to a local vocabulary. Although the rdfs:subClassOf or rdfs:subPropertyOf properties can be used 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 property; since both schemas have simply subclassed the original property, the information that these two properties are equivalent is lost. We feel that these features will be necessary because achieving consensus for schema names will be impossible on a world-wide scale.

However, the main weakness of RDF is its limited ability to cope with a rapidly changing and distributed Web. Since there are no inference rules, there is no way to map between different representations of the same concepts. Although RDF does provide 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. This is similar to the SHOE concept of creating a new file for each version of an ontology. However, in RDF, a schema revision is really a schema that extends the original version; its only link to the original schema is by using 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, schemas and resources that refer to the schema that was updated 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. In SHOE, the id as well as the version number of an ontology are explicit.

Like SHOE, RDF realizes that different schemas may use the same strings to represent different conceptual meanings. SHOE solves this with <USE-ONTOLOGY> tags that identify ontologies and specify prefixes, and then requires that the appropriate prefixes are appended to all element references. RDF uses XML namespaces to assign a separate namespace to each schema. There are two disadvantages to this approach. First since namespaces can be applied to any XML DTD, there is no way to know if a particular set of tags is RDF or just intermeshed tags from a different DTD. Second, each RDF section must explicitly specify the namespace for every schema that is referenced in that section, even for schemas that were extended by a schema whose namespace has already been specified. SHOE, on the other hand, allows access to any elements from ontologies extended by the one in use via the prefix chaining mechanism.

6 Future Work

We have learned many lessons from the design, development and use of our two applications. This work leaves us with many interesting research directions.

6.1 Conduct a Systematic Evaluation

The work described in this paper has shown that SHOE can be made to work for small applications using Parka for reasoning and data storage. However, the language still needs to be demonstrated in a large scale internet environment and a comparison should be made of alternate knowledge base systems. We also plan to conduct a systematic evaluation that considers a number of factors, including performance and usability.

An important part of performance is how well the system retrieves information. The standard metrics from the field of information retrieval are precision and recall. Precision is measured as the percentage of retrieved documents that are relevant. Recall is a measured as the percentage of available relevant documents that were retrieved. Unfortunately, with a system like SHOE, these metrics are highly dependent on the quality of the annotation by web site authors. Most web-sites will not have incorrect annotations and there is no ambiguity in the annotations, so SHOE should be able to have 100% precision. If the entire site has been indexed by SHOE, and every page is annotated to the level required, then it is also possible to have 100% recall. Any SHOE evaluation must consider these factors to develop an unbiased metric. In addition, 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. To determine 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?

To support our performance evaluations, we are currently extending the number of annotated pages used by both of the applications described in this paper. We are adding more detailed annotations to the 8282 computer science department pages used by the WebKB project and plan to add information from more universities as well. We are also working with JIFSAN to get a large mass of pages annotated for the TSE application. These efforts will result in datasets that are large enough to conduct truly meaningful evaluations.

6.2 Improve Usability of Tools

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 unrealistic at the time. However, lightweight NLP techniques and semi-automatic annotation may prove useful. To integrate such methods into SHOE without building special purpose tools for each domain, we could specify in our ontologies patterns that indicate relationships or text that serves as evidence that a particular concept is present.

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 [12]. 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. 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.

6.3 Experiment with Inference Techniques

One of the main challenges of an internet-based reasoning system is to scale well enough to handle the volume of data. Due to the similarity between SHOE's semantics and that of datalog, recent results in the study of the later can be used to implement efficient SHOE inference engines. However, the Web's size is still overwhelming. Fortunately, we may be able to relax some requirements for inference. Since it is unlikely that any reasoning system will have access to all information available on the Web, its knowledge base will by necessity be incomplete, and thus even a complete reasoning procedure will be "incomplete" in terms of the facts that can be derived from the entire Web. As such, using an incomplete inference algorithm may not be so bad; one that limits the depth of inference, the time of inference, or the total number of inferences could be useful in keeping the execution time to reasonable levels. Furthermore, it is likely that the most important facts will be those that someone has taken the time to explicitly annotate, and a means to rank results depending on the number of required inferences may help to determine the significance of an answer. An IDA backward-chaining search or a forward-chaining search could return results in the desired order. We will experiment with various inference strategies to determine those that are most applicable to this problem area.

6.4 Explore Language Issues

Our demonstration applications made us aware of issues that are related to the design of the language. A well known problem in knowledge representation systems is that an instance from one perspective can often be considered a category from another. For this reason, page authors have to carefully consider whether the object they are talking about is really an individual entity that will not be subdivided at later date or if it is a class that was just omitted from the ontology. Although it is possible to create a new ontology version with the new class at any time, all references to the original instance will have to be changed. Since this is likely to occur frequently, we are considering adding a language feature that can state that a category is equivalent to an instance.

The TSE application also proved that arithmetic functions would be a very useful feature in the language. JIFSAN wants the TSE Risk Website to be a tool to assist risk assessors, and has asked for the capability to perform automatic risk calculations in SHOE. A sample query might be "What is the risk of a product given the risk of its source materials, the processing of these materials and how each process affects the risk?" To perform this type of inference, we would need to add arithmetic functions to the language. However, if an arithmetic function is used recursively in a rule, inference procedures may never terminate. We are looking into the restricted use of functions in inference rules that have a terminating characteristic and yet are still useful to wide variety of problems.

In the course of building our applications, there were many situations where we would have liked to have an inheritance feature. For example, in the TSE application, most types of rendering have certain properties in common. In the current version of SHOE, there is no way to specify properties at the category level, and thus subcategories and instances have nothing to inherit. However, inference rules can be written that state $Category(x, A) \Rightarrow property(x, y)$ where y is the value of property that is inheritable from class A. It is important to note that this is different from the traditional frame-based or semantic network concept of inheritance, because there is no mechanism for values inherited from more specific classes to override the values from more general classes. However, our looser form of inheritance has the advantage that it makes the system monotonic. We will look to see if the benefits of adding true inheritance to SHOE are worth giving up the property of monotonicity.

6.5 Build a Variety of Applications

In Section 3, we discuss a number of types of systems that could be built using SHOE. So far, we have only built systems that follow one of these approaches: they use a web-crawler that stores all relevant knowledge in a knowledge base. This "knowledge indexing" approach works best for applications with a focused, but still widely distributed, domain. However, SHOE can be used to support general applications to locate information on the entire Web as well. We believe that the other types of systems we discussed provide rich areas for research, especially the on-line search agent. Such agents must choose a good point from which to

begin the search for an answer to the query and then must use their knowledge to help them traverse the Web and locate the requested information efficiently.

7 Conclusion

In this paper, we have argued that knowledge representation is important to the World Wide Web, but that an internet knowledge representation system must not rely on many of the common assumptions in our field. We have described SHOE, a knowledge representation language that makes assumptions that are more suitable to the Internet. In Section 1, we listed a number of characteristics that must be addressed by a web-based KR system; we will now summarize how SHOE addresses these features. First, the system must be scalable: SHOE's first step in this direction is limiting expressivity although there is much research to be done on improving query efficiency. Second, the Web is an "open-world": as a result, SHOE does not allow conclusions to be drawn from lack of information. Third, the Web is dynamic: SHOE addresses the need for rapid change with simple yet powerful extension and versioning mechanisms. With ontologies SHOE addresses heterogeneity, provides structure for information, and provides a context for reasoning about different domains. Finally, SHOE copes with the distributed ownership of data by treating all discovered information as claims instead of facts.

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 a critical evaluation of SHOE (based on a large collection of computer science pages that are available on the web), and the development of tools that make the language more user friendly.

SHOE gives HTML authors an easy but powerful way to encode useful knowledge in Web documents and it offers intelligent agents a much more sophisticated mechanism for knowledge discovery. Such a semantic language could provide context and relation information for more powerful search engines. In addition, SHOE could greatly expand the speed and usefulness of intelligent agents by removing the single most significant barrier to their effectiveness: a need to comprehend text and graphical presentation as people do.

A key goal of this project is to encourage knowledge representation researchers to consider new directions for our field based on the new needs of the information technology revolution. To this end, we have made SHOE, the first language of this kind, freely available on the web, and have made the Java libraries and our prototype tools available as well. 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.

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

[6] 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).

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

[8] 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).

[9] 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).

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

[11] 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).

[12] 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.

[13] D. Fensel, S. Decker, M. Erdmann, and R. Studer, Ontobroker: How to enable intelligent access to the WWW, in: *AAAI-98 Workshop on AI and Information Integration* (Madison, WI, 1998).

[14] J. Heflin, J. Hendler, and S. Luke, Coping with Changing Ontologies in a Distributed Environment, in: *AAAI-99 Workshop on Ontology Management*, (AAAI/MIT Press, 1999).

[15] 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).

[16] R. Kent and C. Neuss, Creating a Web Analysis and Visualization Environment, *Computer Networks and ISDN Systems*, 28 (1995).

[17] 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).

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

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

[20] 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).

[21] 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).

[22] 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).

[23] 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).

[24] 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).

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

[26] 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.

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