The Ontolingua Server:
a Tool for Collaborative Ontology Construction

Adam Farquhar **Richard Fikes** **James Rice**

Knowledge Systems Laboratory
Stanford University
Gates Building 2A MC9020
Stanford, CA 94305
{afarquhar,fikes,rice}@ksl.stanford.edu

Abstract

Reusable ontologies are becoming increasingly important for tasks such as information integration, knowledge-level interoperation, and knowledge-base development. We have developed a set of tools and services to support the process of achieving consensus on common shared ontologies by geographically distributed groups. These tools make use of the world-wide web to enable wide access and provide users with the ability to publish, browse, create, and edit ontologies stored on an *ontology server*. Users can quickly assemble a new ontology from a library of modules. We discuss how our system was constructed, how it exploits existing protocols and browsing tools, and our experience supporting hundreds of users. We describe applications using our tools to achieve consensus on ontologies and to integrate information.

The Ontolingua Server may be accessed through the URL

`http://ontolingua.stanford.edu:5915/`

1. INTRODUCTION

1.1 The Need for Ontologies

In order for an agent to make statements and ask queries about a subject domain, it must use a conceptualization of that domain. A domain conceptualization names and describes the entities that may exist in that domain and the relationships among those entities. It therefore provides a vocabulary for representing and communicating knowledge about the domain.

Explicit specifications of domain conceptualizations, called ontologies, are essential for the development and use of intelligent systems as well as for the interoperation of heterogeneous systems. They provide the system developer with both the vocabulary for representing domain knowledge and a core of domain knowledge (i.e., the descriptions of the vocabulary terms) to be represented. Ontologies inform the system user of the vocabulary that is available for interacting with the system and about the domain and the meaning that the system ascribes to terms in that vocabulary. Ontologies are also crucial for enabling knowledge-level interoperation of agents, since meaningful interaction among agents can occur only when they share a common interpretation of the vocabulary used in their communications. Finally, ontologies are useful in many ways for human understanding and interaction. For example, they can serve as the embodiment of (and reference for) a consensus reached by a professional community (e.g., physicians) on the meaning of a technical vocabulary that is to be used in their interactions (e.g., exchange of patient records).

Ontology construction is difficult and time consuming. This high development cost is a major barrier to the building of large scale intelligent systems and to widespread knowledge-level interactions of computer-based agents. Since many conceptualizations are intended to be useful for a wide variety of tasks, an important means of removing this barrier is to encode ontologies in a reusable form so that large portions of an ontology for a given application can be assembled from existing ontologies in ontology repositories.

We have been working to develop and disseminate effective easy-to-use tools for creating, evaluating, accessing, using, and maintaining reusable ontologies (Fikes et al. 1991). We have developed a set of tools and services to support not only the development of ontologies by individuals, but also the process of achieving consensus on common ontologies by distributed groups. These tools make use of the world-wide web to enable wide access and provide users with the ability to publish, browse, create, and edit ontologies stored on an *ontology server*. The design of the web-based interface and the underlying infrastructure is detailed in (Rice et al. 1996). These tools and services provide many of the facilities that are crucial for promoting the use of ontologies and knowledge-level agent interaction including:

- A semi-formal representation language that supports the description of terms both informally in natural language and formally in a rigorous computer interpretable knowledge representation language. We use an extended version of the Ontolingua language (Gruber 1992) which provides a frame-like syntax in addition to full first order logic as specified in the Knowledge Interchange Format (KIF) (Genesereth 1991). The language supports semi-formal definitions through the use of documentation strings and notes in addition to the formal specifications.
- Browsing and retrieval of ontologies from repositories. Browsing requires presentation of formal descriptions in an easily understood format. We make it easiest to express and browse knowledge that fits into an object-oriented frame view. We believe that the growing popularity of object systems (languages, databases, CORBA, etc.) substantially widens the group of people that are comfortable working in this paradigm. The key, however, is that the presentation of these objects is separated from their internal representation.
- Assembly, customization, and extension of ontologies from repositories. This requires the ability to identify and resolve name conflicts and to augment descriptions of terms from the assembled ontologies. We have extended our Ontolingua language so that users can quickly assemble a new ontology from a library of modules, as well as extend or restrict definitions from the library.
- Facilities for translating ontologies from repositories into typical application environments. We have developed translators into a number of representations including CORBA's IDL (Mowbray and Zahavi 1995), Prolog, CLIPS, LOOM (MacGregor 1990), Epikit (Genesereth 1990), KIF (Genesereth and Fikes 1992).
- Facilities for programmatic access to ontologies so that remote applications have reliable access to up-to-date term definitions. We have defined a network protocol and application program interface (API) to enable remote applications to use an Ontolingua Server to learn about the vocabulary in an ontology and, for example, ask about the relations between terms.
- Support for distributed, collaborative development of consensus ontologies. We have developed a network accessible development environment (e.g., using Mosaic or NetScape) with a rich set of features to support concurrent ontology development such as fine-grained locking mechanisms and analysis of alternative definitions from multiple authors.

Ontology development and use technology will succeed when it becomes commonplace for people in a broad spectrum of communities to build and use ontologies routinely (e.g., as spread sheets are and e-mail is becoming). Another indicator of success will be the availability and widespread use of large-scale repositories of reusable ontologies in diverse disciplines (e.g., software development, database design and maintenance, network-based information retrieval, electronic commerce). These indicators of success should emerge when the technology has progressed sufficiently so that the benefits provided by using ontologies significantly outweigh the costs of developing them.

In Section 2 of this paper we discuss new features of the Ontolingua language that support assembly and reuse of existing ontologies from a repository. Section 3 addresses the design of our ontology editing tools. Section 4 presents empirical evidence regarding the usability of the Ontolingua Server and our experience supporting hundreds of users.

1.2 The Ontolingua Server Architecture

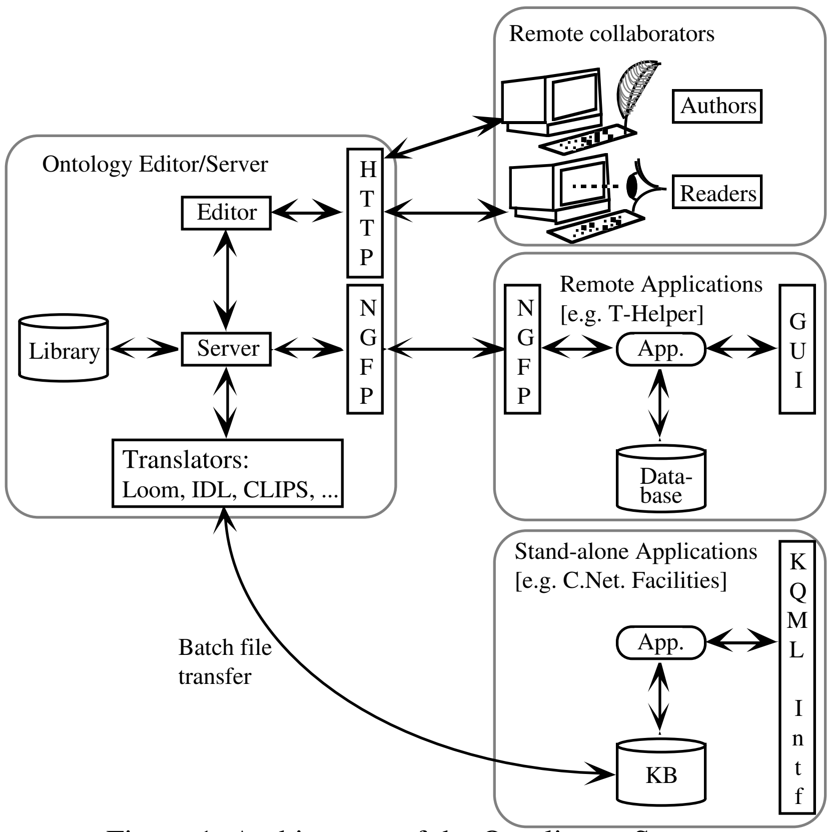

Figure 1: Architecture of the Ontolingua Server.

We have developed and deployed an Ontolingua Server that satisfies many of the requirements described in the previous section. To motivate the design and architecture of the system, consider the problem of constructing and using an ontology for therapeutic drugs (Gennari et al. 1995). Like many ontology projects, this is a substantial and challenging problem. No single expert has the expertise necessary to construct such an ontology. Thus, it is essential for experts to collaborate on construction. Furthermore, it is essential for prospective users of such an ontology to examine, evaluate, and critique it. These prospective users will be geographically distributed across a variety of organizations (e.g., hospitals, insurers, government agencies). Once this ontology has been built, it can be used in a number of ways. First, practitioners who want to be sure of a term's meaning can use it as a reference. Second, networked programmatic agents such as medical expert systems can query an Ontolingua Server at run-time. For instance, the T-Helper system (Gennari et al. 1995) can query an Ontolingua Server to determine the characteristics of therapeutic drugs (e.g., is zidovine an anti-retroviral drug?). Third, standalone programs can use ontologies as a skeleton for their own representations. In addition to knowledge-based systems, it would be valuable to provide a starting point for constructing database schema, and object-oriented programs.

The design and architecture of the Ontolingua Server supports these modes of use. Figure 1 shows a schematic view of the system. The leftmost box depicts the general-purpose Ontolingua editor and server. The server provides access to a library of ontologies, allows new ontologies to be created, and existing ontologies to be modified. There are three primary modes of interaction with the Ontolingua Server indicated by the three boxes on the right side of Figure 1.

First, remote distributed groups can use their web browsers to browse, build, and maintain ontologies stored at the server. The server interacts with them through the now widespread hypertext transfer protocol (HTTP) and hypertext language (HTML) used on the world wide web. This makes the server accessible by a very wide audience. Any user familiar with common web browsing tools such as NetScape Navigator™ or NCSA's Mosaic can browse, build, and maintain ontologies stored at the server. The server allows multiple users to work simultaneously on an ontology in a shared *session*. The editor provides a number of features to support collaborative work (e.g., notifications, comparison, shared change logs). The design of the web-based interface and the underlying infrastructure is detailed in (Rice et al. 1996).

Second, remote applications may query and modify ontologies stored at the server over the Internet. These programmatic interactions use a network API that extends the Generic Frame Protocol (Karp, Myers, and Gruber 1995) with a network interface. The network interface supports queries (e.g., is `zidovudine` an `antiretroviral-drug`) as well as updates (e.g., create a subclass of `vector-quantity` called `3d-vector-quantity`).

Third, a user can translate an ontology into a format used by a specific application (Gruber 1993). The resulting translation can be used in a number of ways. For example, a CLIPS translation produces a set of class definitions and inference rules that can run directly in a CLIPS-based application. An Interface Definition Language (IDL) translation produces an IDL header file that a CORBA compliant program can use to interact with an Object Request Broker (ORB). A KIF translation produces a file of logical sentences that can be used by a logic-based facilitator, such as the one fielded by CommerceNet to draw inferences in response to client queries and to route these queries correctly.

2. THE ONTOLINGUA LANGUAGE

The original Ontolingua language, as described in (Gruber 1993), was designed to support the design and specification of ontologies with a clear logical semantics. To accomplish this, Gruber started from KIF. KIF is a monotonic first order logic with a simple syntax and some minor extensions to support reasoning about relations. Gruber extended KIF with additional syntax to capture intuitive bundling of axioms into definitional forms with ontological significance; and a *Frame Ontology* to define object-oriented and frame-language terms. The Ontolingua Server has extended the original language in two ways. First, it provides explicit support for building ontological modules that can be assembled, extended, and refined in a new ontology. Second, it makes an explicit separation between an ontology's *presentation* and *representation.*

The original Ontolingua language provided limited support for defining ontological modules in the form of a tree of named ontologies. As we shall see in Section 2.1, our users found this simple model to be inadequate in several ways. Furthermore, it did not have a clearly articulated semantics; this was in sharp conflict with the basic goals of the language. We have subsequently introduced a new inclusion model which makes a clear separation between the set of axioms in an ontology and the input-output behavior of an implementation that must translate sequences of characters into axioms.

The separation of presentation and representation has always been implicit in Ontolingua's translation approach to sharing ontologies. In the current system, however, the explicit recognition of this distinction has become a key notion. The representation, the underlying meaning, of an ontology is always defined by a set of KIF axioms. In Ontolingua, the representation is always simple, clear, and unambiguous. The presentation is the manner in which these KIF axioms are viewed and manipulated by a user. Designing a good presentation involves user expectations and assumptions. It may even be complex, murky, redundant, and ambiguous — if that is what the user wants or what the semantics of a translation target demands.

As we shall see in Section 3, the presentation in the Ontolingua Server's browsing and editing environment, in particular, is tailored for object-oriented or frame-language descriptions of the world. We hope that users find the presentation clear, but we guarantee that each statement corresponds unambiguously to a KIF axiom. The vocabulary used in this presentation is defined in the Frame Ontology. The Frame Ontology defines terms including class, subclass-of, slot, slot-value-type, slot-cardinality, facet, and so on. If an ontology is defined using this vocabulary, the Ontolingua Server can present it in a form that many users find quite palatable.

A key property of the extended Ontolingua Language and its presentation in the Ontolingua Server is that axioms that do *not* fit into the frame language *are allowed*. There is no restriction on expressiveness. This is extremely important for an ontology development environment. In contrast with an inference tool or a traditional knowledge representation tool for which tractability is paramount, an ontology development tool must support expression. For example, if a user wishes to state the disjunction that a pass grade is equivalent to an A, B, or C, an ontology development environment must allow her to state it.

The Ontolingua Server, however, must operate on ontologies and, for instance, translate them into less expressive languages. For this reason, the editing environment *encourages* users to stay within the relatively simple frame sublanguage. Commands for creating subclasses, adding slots, constraining slot values, and so on, are easy to find and use. The frame-language axioms are presented simply and concisely. It is *possible*, however, for users to write arbitrary KIF axioms. Even if, in the extreme case, an axiom is untranslatable, it will still server as an important formal specification of the authors intention. Indeed, because KIF, and consequently Ontolingua, is monotonic, performing weaker translations will still retain their correctness.

To summarize, the underlying representation for an ontology is a set of KIF axioms. These sentences are projected through a variety of lenses to produce the editor's frame pages, HTML documents, LOOM knowledge bases, Prolog clauses, and objects that can be manipulated using the (N)GFP. A rich infrastructure, not described in this paper, allows a new projection to be assembled from components.

2.1 Assembling an Ontology

We want ontologies to be practical and useful artifacts. This means that the effort required to construct new ontologies must be minimized and the overall effort required to construct an ontology must be amortized over multiple uses and users. We enable ontology writers to reuse existing ontologies in a flexible and powerful way. In this section, we show how the Ontolingua Server allows users to reuse existing ontologies from a modular structured library by inclusion, polymorphic refinement, and restriction.

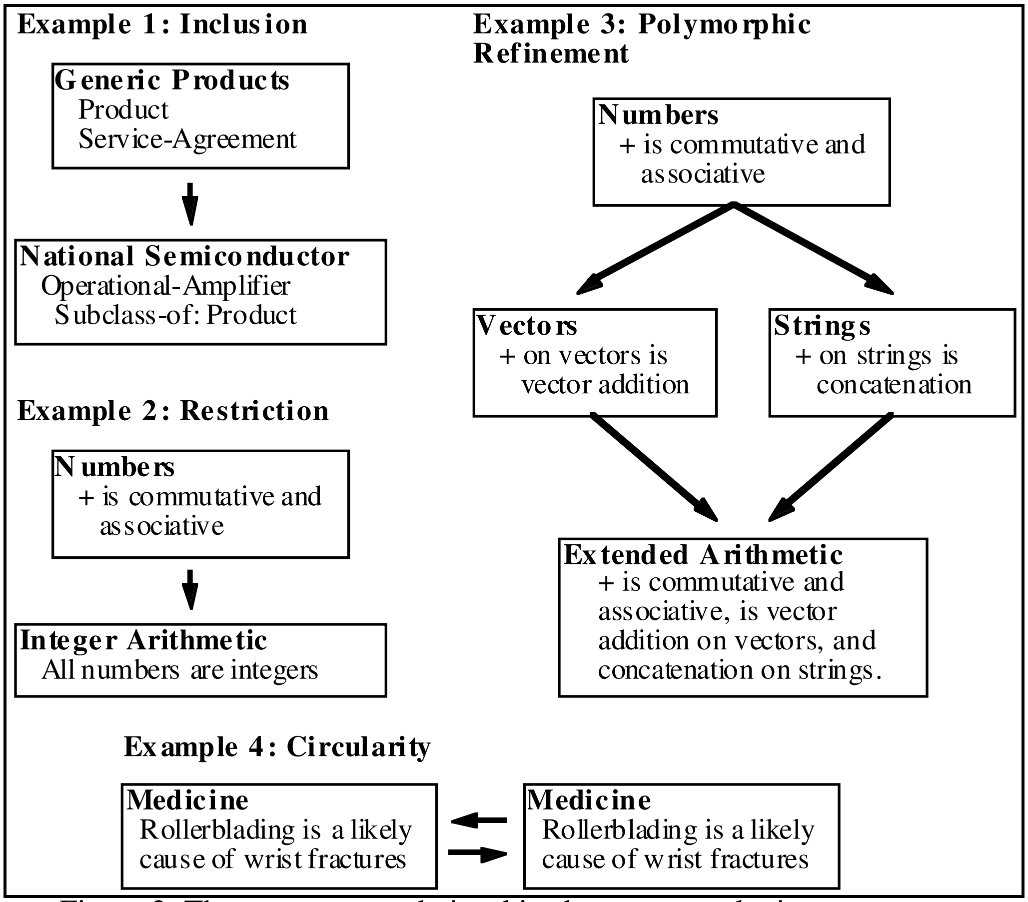

Figure 2: There are many relationships between ontologies.

Formally, ontologies in our system are specifications of axiomatic logical theories. An ontology specification consists of a vocabulary of non-logical symbols and a set of KIF axioms that constrain the acceptable referents of those symbols. An Ontolingua user specifies these axioms in the form of definitions of classes, relations, functions, and constants. *Non-logical symbols* are the atoms that name relations, functions, and constants. In this section, we describe our approach to providing ontology reuse in terms of operations on sets of axioms and the non-logical symbols that occur in them.

Figure 2 shows several motivating examples that are drawn from our ontology building experience. Example 1 shows the simplest relation between ontologies: *inclusion*. The author of a National Semiconductor product ontology needs to represent basic information about products, prices, services, and so on. She wants to include the entire contents of the generic product ontology from the ontology library without modification.[1] In Example 2, the author wishes to extend the addition operator + in two distinct ways. The library contains axioms about the addition operator in the KIF-numbers ontology (e.g., it is associative, commutative, has 0 as an identity element, etc.). She wishes to extend the addition operator to apply to vectors in one ontology and to apply to strings in another ontology — we refer to this operation as *polymorphic refinement*. In Example 3, we see that the inclusion relations between ontologies may be circular. We consider two ontologies: one for medicine and another for sports. The medical ontology will need to refer to

[1]Notice that by "inclusion" here, we do not mean "cut and paste the contents of the product ontology into the HP ontology file". This interpretation would result in unfortunate version dependencies.

a variety of terms from the sports ontology (e.g., roller-blading is a leading cause of wrist fractures in teens) and the sports ontology must also refer to medical terms (e.g., weight-lifters may use anabolic steroids to increase muscle growth). We must handle this sort of relationship carefully because the ontology designers do not want either ontology to be polluted by the non-logical symbols from the other. In Example 4, we see that specialized ontologies may make simplifying assumptions that *restrict* the included axioms. For example, in the integer-arithmetic ontology, all numbers are restricted to be integers.

Many knowledge representation systems have addressed these issues in one way or another. Before turning to our solution, we will discuss some of the approaches that others have used, illustrate some of their shortcomings, and use them to motivate our novel design choices.

The easiest and simplest approach is to provide no explicit support for modularizing represented knowledge — let the author beware. For instance, the THEO system (Mitchell et al. 1989) uses a single knowledge base and a single set of axioms. In some sense, this enables Examples 1-3 to be represented, but it has two key drawbacks: First, it is impossible to restrict definitions (Example 4). Second, by eliminating modularity, it makes understanding and evaluating ontologies a nightmare. Authors using systems like this often resort to lexical conventions to discriminate between symbols (e.g., `+@kif-numbers`, `+@vectors`, `+@strings`). Without automated support, such conventions are difficult to enforce. Furthermore, enforcing them may not even be desirable. In Example 2, the axioms in the `vectors` ontology are about the same + operator as the axioms in `kif-numbers`.

A fairly common extension is to allow a directed acyclic graph (DAG) of inclusion relations between "theories" such as provided by Genesereth's Epikit. That mechanism supports modularity, restrictions, and incompatible augmentations. It has two drawbacks: First, no cycles are allowed among theories. As we have seen, it is both natural and desirable to have cyclic relationships between terms in ontologies.[2] Second, in its simple form, this mechanism results in unnecessary name conflicts. For instance, an ontology for scientific course work might include ontologies for chemistry and academics, both of which define `tests`, but in different ways. There must be a way of discriminating between `tests` in chemistry and `tests` in academics.

The LOOM system provides a DAG of inclusion relationships, but extends the simple approach by allowing references to non-logical symbols in ontologies that have not been included. Referencing a symbol in an unincluded ontology, however, does not include all of the axioms from that ontology, but only minimal type information. This conflates the declarative semantics, as defined by the axioms, with pragmatic information about which axioms to apply during problem solving.

There are two distinct aspects to our solution: (1) the *inclusion model* which defines how axioms and non-logical symbols are included in new ontologies, and (2) the *interface model* which defines the relationship between character strings input by (or output to) the user and non-logical symbols in the ontologies.

[2]Indeed, we initially wanted to avoid the additional complexity introduced by allowing circular references, but our users demanded it. For any particular example in which circular references occur, it is always possible to create a new ontology, (e.g., sports-medicine) that contains the subset of the ontologies in the cycle. This is not a practical solution, however, because it may require the entire structure of the ontology library to be changed to add a single axiom. This would make it impossible to have a general-purpose library of ontological fragments that can be reused.

2.2 A Semantic Model for Ontology Inclusion

In order to facilitate the reuse of existing ontologies, the Ontolingua Server provides a facility for *including* one ontology in another as follows. Each ontology is considered to be specified by a vocabulary of non-logical symbols and a set of axioms. Formally, including an ontology `A` in an ontology `B` requires specifying a translation of the vocabulary of `A` into the vocabulary of `B`, applying that translation to the *axioms of* `A`, and adding the translated axioms to the *axioms in the specification of* `B`. We say that the axioms in the resulting set are "the axioms of ontology `B`" so that if `B` is later included in some other ontology `C`, the ontology `C` will include translated versions of both the axioms in the specification of `B` and the axioms of `A` . Thus, when we say "the axioms of ontology `O`", we mean the union of the "axioms in the specification of `O`" and the axioms of any ontology included by `O`. This notion of inclusion defines a directed graph of inclusion relationships that can contain cycles. We allow ontology inclusion to be transitive and say that ontology A is included in ontology `B` if there is a path in the ontology inclusion graph from `A` to `B` .

The Ontolingua Server eliminates symbol conflicts among ontologies in its internal representation by making the symbol vocabulary of every ontology disjoint from the symbol vocabulary of all other ontologies. That is, in the internal representation, the symbol `S` in the vocabulary of ontology `A` is a different symbol from the symbol `S` in ontology `B`.

Given that symbol vocabularies are disjoint, the Ontolingua Server can assume in its internal representation that the translation used in all inclusion relationships is the identity translation. Therefore, in the internal representation, including an ontology `A` in an ontology B simply means adding the axioms of `A` to the axioms of `B` .

Note that in this model of ontology inclusion, cyclic inclusion graphs are not a problem since the only effect of ontology inclusion is the unioning of sets of axioms.

If an ontology `A` contains an axiom that references a symbol in the vocabulary of an ontology `B`, then `B` is considered to be included in `A`. the Ontolingua Server allows users to state explicit inclusion relationships between ontologies and implicitly creates inclusion relationships based on symbol references in axioms.

2.3 Natural Input and Output

"... no more complex than the Common Lisp package system"
— James Rice

The semantic model introduced above provides a powerful, simple, and unambiguous way for ontologies to be assembled and reused. However, in order to eliminate ambiguity, it requires symbols to be given clumsy extended unique names that may be unknown to the user. Moreover, it does not allow a user to perform important operations such as renaming symbols from included ontologies or selectively controlling which symbols are to be imported from included ontologies or exported to other ontologies. The Ontolingua Server solves these problems with additional capabilities that are a part of its facilities for converting symbol references in input/output text to and from the internal symbol representation.

Ontolingua requires that any non-logical symbol referred to in an input or output stream be defined in some ontology and be assigned a name. The ontology in which a symbol is defined is called that symbol's home ontology. Similarly, each ontology has a name that uniquely distinguishes it from any other ontology.

The Ontolingua Server interprets a symbol reference in an input stream or produces a symbol reference in an output stream from the perspective of a given ontology. For example, if the symbol S is defined in ontology A and also defined in ontology B, then from the perspective of ontology A, the input text "S" is interpreted as “the symbol named S defined in ontology A”, whereas from the perspective of ontology B, the input text "S" is interpreted as "the symbol named S defined in ontology B".

The perspective from which any given symbol reference is to be interpreted can be explicitly specified by attaching a suffix to the symbol name consisting of the character "@" following by the name of an ontology. So, for example, the symbol named S interpreted from the perspective of ontology A can be explicitly referred to as "S@A". A symbol reference that includes the @«ontology name» suffix is said to be a fully qualified reference. Fully qualified references enable symbols defined in any ontology to be referenced in any other ontology.

The Ontolingua Server input/output system provides a symbol renaming facility that allows a user to assign a name to a symbol which is local to the perspective of a given ontology. A renaming is specified by a rule that includes an ontology name, a symbol reference, and a name that is to be used as the local name of the given symbol from the perspective of the given ontology. Given such a renaming rule, the system will recognize the local name as a reference to the given symbol when processing input in the given perspective, and will use the local name to refer to the given symbol when producing output from the given perspective. So, for example, a renaming rule might specify that in ontology A, the local name for auto@vehicles is to be car. This facility enables an ontology developer to refer to symbols from other ontologies using names that are appropriate to a given ontology and to specify how naming conflicts among symbols from multiple ontologies are to be resolved.

For convenience of input and parsimony of output, the @«ontology name» suffix can be omitted from symbol references when the symbol name itself is unambiguous from the intended perspective. In particular, a given symbol can be referred to from the perspective of an ontology A simply as S if and only if the given symbol:

- Is defined in ontology A with name S;
- Has been renamed to S in ontology A; or
- Has been *imported* into ontology A.

A name that can be used to refer to a defined symbol from the perspective of a given ontology is said to be *recognized* in that ontology. Thus, the convention given above for omitting the @«ontology name» suffix from symbol references implies that a name S is recognized in an ontology A if and only if S is the name of a symbol defined in A, or S is the name of a symbol that has been imported into A and has not been renamed in A, or S is the local name for a symbol in A.

We now describe the mechanisms for importing symbols into an ontology.

Each defined symbol is designated as being *public* or as being *private* to the ontology in which it is defined. The system considers symbols to be public by default so that users can ignore the public/private distinction until they encounter or want to define private symbols.[3]

The Ontolingua Server associates with each ontology a set of ontologies whose public symbols are imported into the ontology. User commands are available for editing that set of ontologies. However, in order to simplify the use of this symbol importing mecha-

[3]Users can change the default on a per-ontology basis.

nism, by default the Ontolingua Server automatically adds to this set any ontology that is explicitly included. Thus, users can ignore the distinction between symbol importing and explicit inclusion until they want to override that default.

In order to provide control over the importing of individual symbols, the Ontolingua Server associates with each ontology a set of *shadowed* symbols that are blocked from being imported into the ontology. That set of shadowed symbols overrides symbols in the set of imported ontologies in that a symbol is imported into an ontology `A` only if the symbol is a public symbol defined in an ontology that `A` imports and is not shadowed in `A`, or the symbol is a public symbol in an ontology that does not import, but is imported by means of an identity renaming (e.g., `S@A` goes to `S@B`)

The Ontolingua Server uses the shadowing mechanism to prevent ambiguities from occurring in the text form of symbol references by automatically blocking the importation into an ontology of any symbols which would have the same text form in that ontology's perspective. Thus, if there is a symbol that can be referred to by `S` from the perspective of ontology `A`, a different symbol that can be referred to by `S` from the perspective of ontology `B`, and the public symbols from both ontologies `A` and `B` are imported into ontology `C`, then the Ontolingua Server automatically adds `S@A` and `S@B` to C's list of shadowed symbols in order to prevent "`S`" from being an ambiguous symbol reference from the perspective of ontology `C`. This automatic shadowing occurs irrespective of the order in which the definitions and inclusion relationships are specified. So, for example, if the inclusion relationship between ontologies `A` and `C` already exists when a symbol that can be referred to by `S@A` is defined, then `S` will be added to the set of symbols shadowed in ontology `C` at the time `S@A` is defined.

2.4 Summary

To summarize, we note how Ontolingua supports ontology inclusion, circular dependencies, and polymorphic refinement by reconsidering the examples from Figure 2.

Using Ontolingua, the ontology inclusion relationship in Example 1 would be explicitly established by the developer of the National Semiconductor Products ontology either as part of the definition of that ontology or as an editing operation after the ontology has been defined. Given that inclusion relationship, public symbols from the Generic Products ontology, such as `Service-Agreement`, whose names do not conflict with other recognized names in the perspective of the National Semiconductor ontology would, by default, be imported into the National Semiconductor Products ontology and therefore could be referred to from the perspective of that ontology by their names (e.g., `Service-Agreement`) without the `@GenericProducts` suffix.

The circular dependencies in the `Medicine` and `Sports` ontologies of Example 2 would be established and presented by using fully qualified names to refer to symbols from the perspective of the other ontology. For example, in the `Medicine` ontology, roller-blading would be referred to as `roller-blading@sports`, and in the `Sports` ontology, steroid tests would be referred to as `steroid-tests@medicine`. The reference to roller-blading in the `Medicine` ontology will cause the axioms of the home ontology of the symbol `roller-blading@sports` to be implicitly included in the `Medicine` ontology, but would not cause the public symbols from the `Sports` ontology to be imported into the `Medicine` ontology.

The polymorphic refinement of + in Example 3 is a case in which some of the subtle properties of implicit ontology inclusion become apparent. If ontology `X` does not explicitly include the `vectors`, `strings`, or `numbers` ontologies, then references to `+@vectors` and `+@strings` in X will cause `numbers` to be implicitly included in `X`, but will not cause `Vectors` or `Strings` to be included since both `+@vectors` and `+@strings` refer to a symbol whose home ontology is `Numbers`. If the vector addition axioms of ontology `Vectors` and/or the string addition axioms of ontology `Strings` are to be included in ontology `X`, then the user must state the inclusion relationship explicitly.

3. THE ONTOLINGUA DEVELOPMENT ENVIRONMENT

Our goal was to create a general environment to facilitate the development and sharing of ontologies. Such an environment must assist the user in the basic development tasks of browsing, creating, maintaining, sharing, and using ontologies. We also realized that many of our users desire to develop ontologies through a consensus process; therefore, we also needed to provides tools to help people collaborate during the development of their ontologies. In this section, we will discuss the features implemented in our web-based ontology environment which provide assistance with basic development tasks, facilitate collaboration, and improve ease-of-use.

3.1 Browsing Ontologies

An essential component of our browsing environment is being able to quickly jump from one term in the ontology to another using hyperlinks. Selecting the name of a term takes the user to a page displaying the definition of that term. The definition consists of both informal documentation and formal statements about that term. Rather than displaying this information in a purely logic-based form, we display this information in an object-oriented or frame-based form (see Figure 3). In a frame, slots are displayed along the left

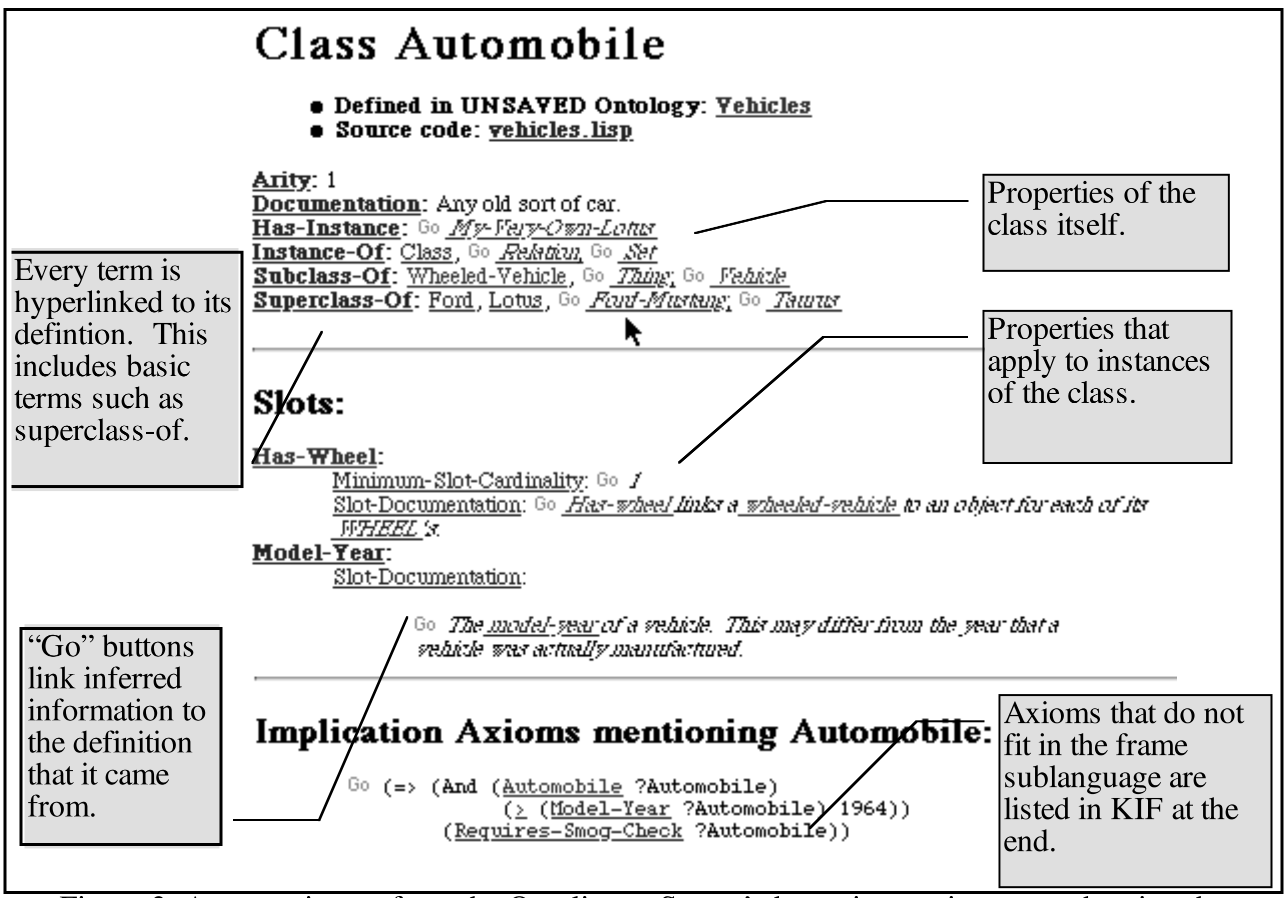

Figure 3: A screen image from the Ontolingua Server's browsing environment showing the class definition `Automobile` in the `vehicles` ontology.

side of the screen and values corresponding to the slots are displayed following the name of the slot. Information that can't be displayed as slot or facet values appears later in the page as axioms.

When users examine an entire ontology, they often want to see information about a term that is inferable from the definition of other terms. To support this feature, the Ontolingua Server performs the limited set of inferences are typically provided by frame-based representation systems. These include inheritance from classes to subclasses and instances, inverses, limited transitivity, and so on. Information that was directly asserted is visually distinguished from inferred information. The direct assertions appear in a roman font and inferred information appears in italics. The user can see where the inferred information was directly asserted by pressing a button labeled "GO" attached to each piece of inferred information.

The Ontolingua Server provides class/subclass browsers that display an entire hierarchy in a compact fashion to give the user a quick overview of an ontology. Rather than using a node and arc graphic display, the hierarchy is displayed in an indented text form with widgets that allow subtrees to be closed or opened (see Figure 4). The class browser interacts with the ontology inclusion graph. The user can select the set of ontologies whose contents should be included in the hierarchy. The user can also focus the display on a single class to reduce clutter in large ontologies. For moderate sized ontologies (150-1000 definitions), we find the indented text format preferable to a graphic display. Furthermore, the textual display substantially reduces the number of bytes required to de-

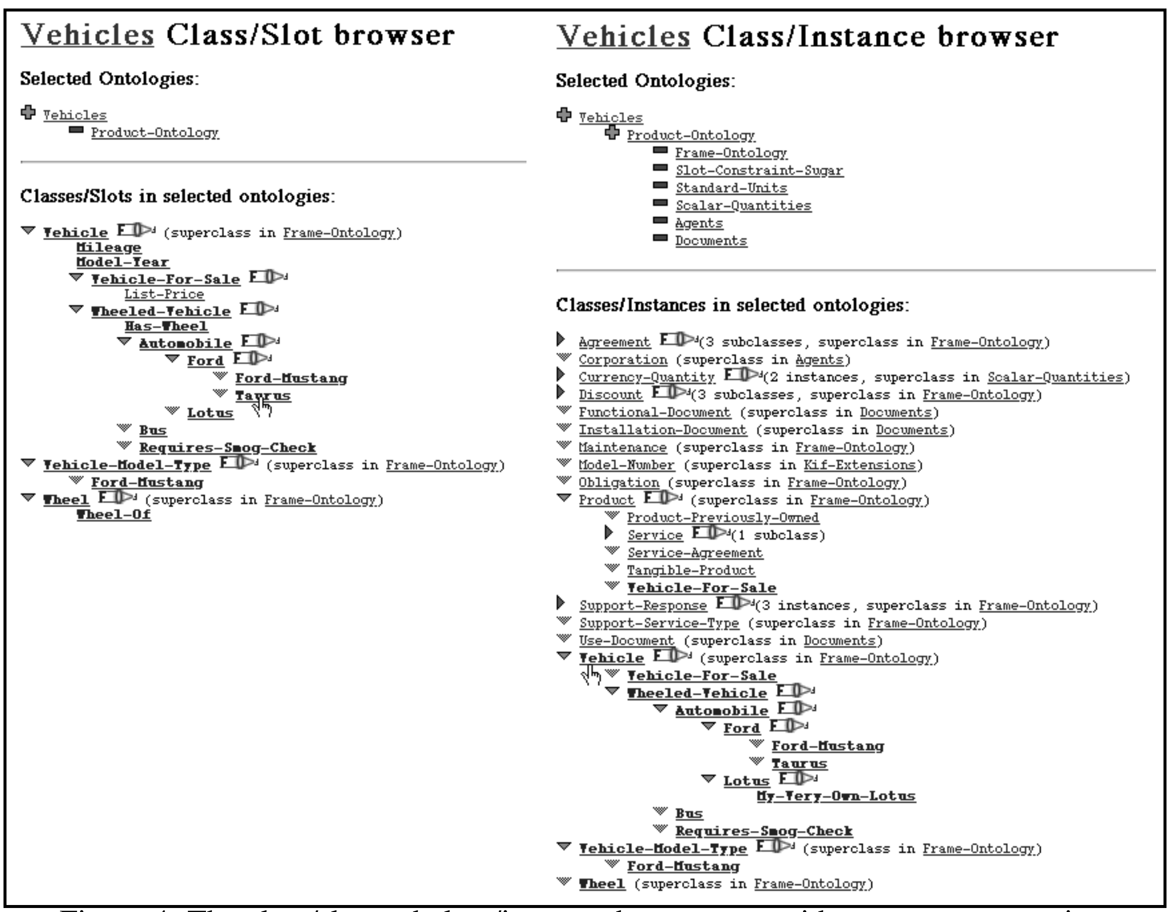

Figure 4: The class/slot and class/instance browsers provide a compact overview of set of ontologies. Only objects in the selected ontologies are displayed.

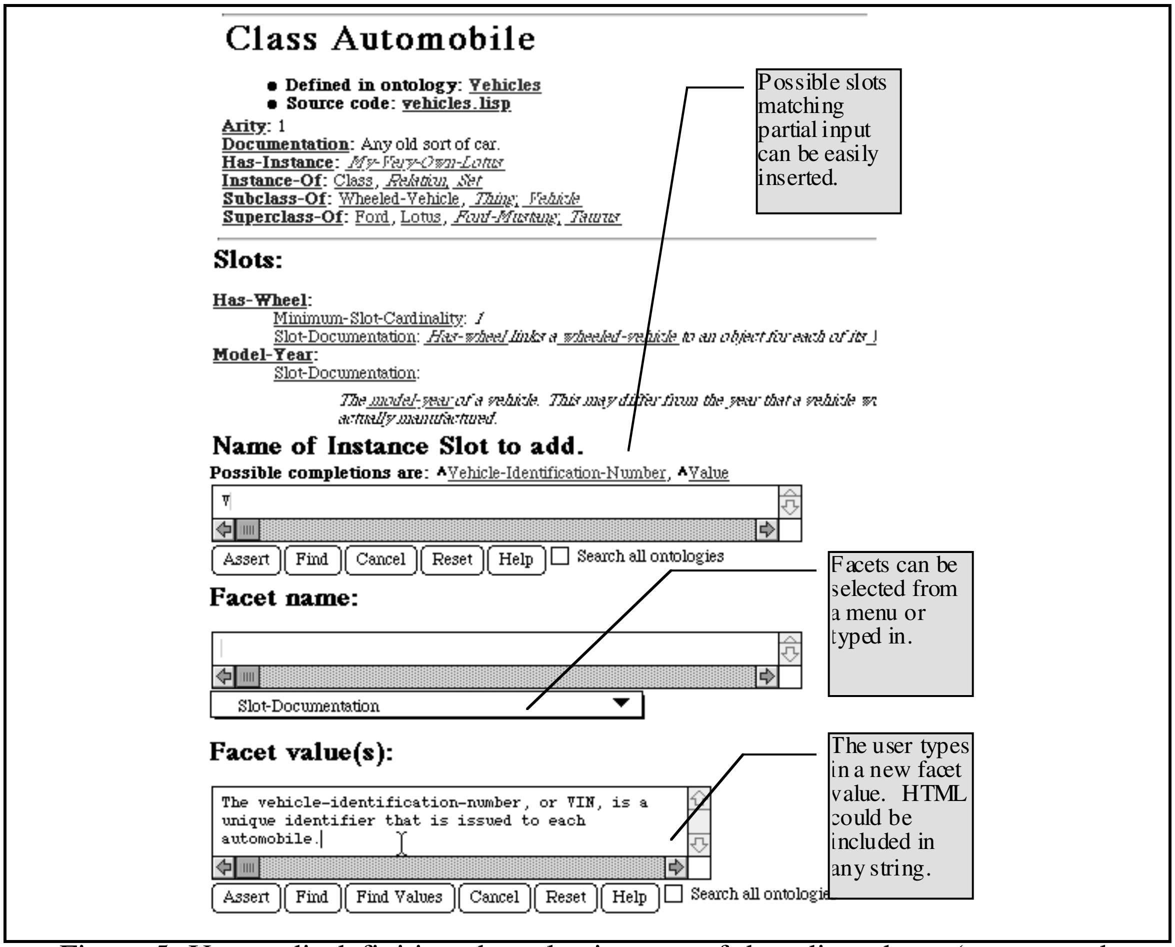

Figure 5: Users edit definitions by selecting one of the edit gadgets (pens, or addition icons). An input field is snipped into the page at the appropriate location. The appearance of the editing and browsing modes are as similar as possible.

scribe the hierarchy. This results in improved performance for our networked tools.

3.2 Building Ontologies

We designed the ontology editing environment to have a similar appearance to the ontology browsing environment so that the user would not need to learn two different interfaces. The difference between the editing environment and the browsing environment is that two new types of icons appear in the editing environment: edit pens and insertion icons. Edit pens appear in front of information that can be modified by the user. When the user wishes to change that information, she selects the appropriate edit pen, and a form for modifying the current information appears. Insertion icons appear wherever a user is allowed to add information such as a new value or facet for a slot. When the user selects one of the insertion icons, a form for entering new information appears. The contents of these forms depends on the type of thing which is being added or edited.

Commands which don't fit into this paradigm appear as menu options at the top of the screen. For example, commands for creating new terms appear on these menus. The user can select which type of term to create, and an appropriate form will prompt the user for information about the new term.

3.2.1 Maintaining Ontologies

The Ontolingua Server provides several features to assist with ontology maintenance. Users can compare two ontologies and see the set of actions that would transform one into the other. This is an easy way to monitor changes in ontology or undo modifications. Users can analyze an ontology for inconsistencies and undefined terms. All slots, slot values, facets, and facet values are checked to make sure that they conform to known constraints (e.g., domain, range, slot value type, and cardinality constraints). During the course of development, users often want to split an ontology into a set of more easily understood, more tightly focused modules. The server provides direct support for splitting a large ontology into several smaller ontologies that may include each other.

Informal documentation is an important part of making ontologies maintainable. The Ontolingua Server supports special key words within the informal documentation known as notes. A user can assign a keyword such as example, verified-by, see-also, modification-by, and formerly-named to provide more structure to the informal documentation.

3.2.2 Sharing Ontologies

The Ontolingua Server provides a number of features to promote the sharing of ontologies. The primary mechanism for supporting ontology reuse is through the library of ontologies which acts as a central repository for reusable ontologies. When someone has an ontology that they believe is ready for others to use, they can choose to publish their ontology in the Ontolingua Server. After the ontology has been approved, it becomes available to anyone with access to the Ontolingua Server through the library.

The Ontolingua Server also provides several tools for searching for terms within ontologies in the library. A user may choose the general search facility which allows them to use wild-cards in searching the entire library for terms whose name matches the specified pattern. Context-sensitive searching is also available whenever the user needs to fill in the name of a term such as when adding a value to a slot. In context-sensitive searching, constraints on which terms are appropriate are used to limit the search.

3.2.3 Collaboratively Developing Ontologies

One of the features which clearly distinguishes the Ontolingua Server from other ontology development environments is its support for distributed collaborative ontology construction. Distributed access is provided by the web interface. Collaborative work is facilitated by (1) user and group access control, and (2) multi-user sessions.

The Ontolingua Server uses a notion of users and groups that is typical in most multi-user file systems. As with file systems, read and write access to ontologies is controlled by the ontology owner giving access to specific groups. This mechanism supports both access protection as well as collaboration across groups of people who are defined within the ontology development environment.

The server provides support for simulataneous work through group sessions. When a suer opens a session, she may assign a group ownership to it. This enables any other members of that group to join the session and work simulataneously on the same set of ontologies. A notification mechanism informs each user of hte changes that other users have made. Notifications are hyperlinked to the changed definitions and describe changes in terms of basic operations such as add, delete, and modify (e.g., “Farquhar added the slot motor-of to the class vehicle). The synchronous nature of the web protocols makes this sort of notification somewhat clumsy — the Server cannot notify users that a change has occurred until they visit a new page. Recent advances in client-side tools, however, are relaxing these restrictions.

3.2.4 Ease of Use

In designing the interface to the Ontolingua Server, we wanted to make a tool which would be simple for a novice to understand and use yet be powerful enough to support experienced users. To accommodate such differing levels of users, we added a large variety of user preferences for controlling the behavior of the user interface.

To make the interface simple, the Ontolingua Server provides four basic types of pages: the table of contents for the ontology library, ontology summary pages, frame pages (for classes, relations or instances), and the class browser. The Ontolingua Server provides a wide variety of other features to make the environment easy to use. The hyperlinked environment makes it easy to provide tutorials, on-line documentation, and context-sensitive help that can be selected at any time. In addition, the Server allows the user to undo or redo any number of modifications they made to the ontology since they last saved it.

3.3 Using Ontologies

Although the Ontolingua Server currently does not provide much inferential power, it does provide some support for using ontologies. One way to use ontologies developed with the Ontolingua Server is to translate the ontology into the representation language of another system such as CLIPS, LOOM, or Prolog. Currently, the Ontolingua Server can translate into ten different representations. Users can then transfer the translation via e-mail. Users may also e-mail their ontologies as standalone hyperwebs, source code, or formatted text.

4. RESULTS

In this section we discuss some of our experience in using and providing the Ontolingua Server over the web. Our general approach of providing access to research software over the web appears to be highly successful. It has increased the numbers of users and simultaneously reduced our development costs by more than an order of magnitude. We have been able to reach a wide audience in a cost effective manner and provide them with a high quality, robust, and useful tool with far higher functionality than would have been possible had we been distributing our software in a conventional fashion.

Research organizations such as KSL have limited resources to devote to maintaining, documenting, and distributing software. In general, we would rather invest our energies in adding new functionality than in porting to yet one more platform or idiosyncratic hardware and software configuration. Furthermore, the users of research software would rather invest their time and energy evaluating and using it than in downloading, configuring, and compiling. We also save them from buying the necessary software licenses and hardware platform needed to support our software.

There have been numerous other benefits: users who are already familiar with their web browsers can use the Ontolingua Server right away. The constrained interface afforded through HTML has actually had positive effects on our interface design — it is much cleaner and more streamlined than it might otherwise have been. Useful context-sensitive hypertext documentation is accessible by any user through the same web interface. The custom HTTP server was easily extended to handle other network protocols such as the Network GFP. The modular state and session organization was easily extended to support several other network services, that are not described here (but are also available through `http://ksl-web.stanford.edu`).

Evaluating the impact that the Ontolingua Server has had on specific projects is rather difficult. We have not yet engaged in any formal study and the logging information that we have kept has been limited. This is partly by design and partly due to the nature of the system. We want users to feel confident about the privacy of their data and interactions with the server. This is especially important for industrial users. It is also not yet clear what sorts of information would be useful to log. Most web servers maintain a complete log of all URLs that are requested from them. This allows them to determine usage patterns such as how often specific documents are examined. In the Ontolingua Server, however, things are not so simple. The URLs that go in and out of the Ontolingua Server encode each request with an identifier that is unique to the command, user, and session. A record of these identifiers would be useless (like recording hash keys for hash tables that have subsequently been cleared). Furthermore, the pages that users access do not come from any fixed set of files on a server. Users create, modify, and delete new objects while they work. In this effectively infinite space of objects, there is relatively little overlap between users. Thus, recording the objects that were visited might be of little interest. Although it might be interesting to record command executions, the Ontolingua Server supports over 500 distinct commands, and the current command architecture makes it difficult to draw any coherent picture of user activity from this sort of log. An important next step is to determine meaningful instrumentation that will help us evaluate the impact that the editor has on collaborative activities.

Nonetheless, we do have a base level of instrumentation that allows us to draw some inferences about the overall patterns of usage and activity.

The Ontolingua Server has been extremely reliable. Since its public announcement at the start of February 1995, the server has been available 99.89% of the time. Including scheduled downtime for hardware and software upgrades, the total downtime has been typically less than one hour per month. A typical period of uninterrupted service is two weeks, and multi-day sessions are common for our active users. This high level of reliability is essential if we expect remote users to make use of our tools and services.

The Ontolingua Server has reached a wide audience with an acceptable level of continued use. Although we believe that ontologies are becoming increasingly useful and important for a number of purposes, the Ontolingua Server is still a research vehicle that is relevant to a very narrow range of people. Thus, we should not expect usage levels to approach that of very general purpose web sites such as Lycos. Furthermore, announcements of the Ontolingua Server have only been made to fairly focused mailing lists of researchers interested in Ontolingua, Shareable Reusable Knowledge Bases, and Qualitative Reasoning. Nonetheless, as Figure 6 indicates, the number of registered users has grown steadily and currently exceeds 1000.

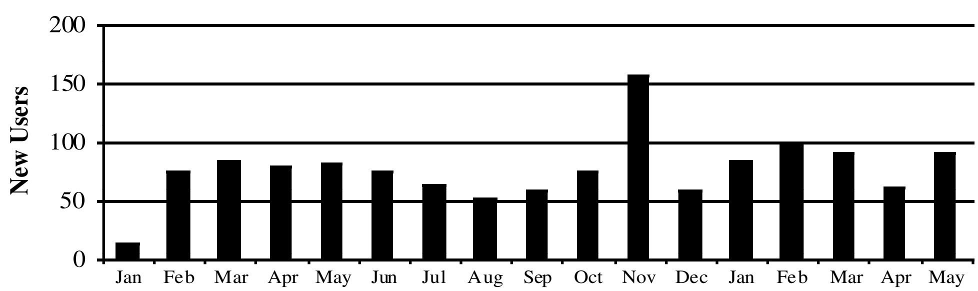

Figure 6: The number of new users registering each month from January 1995 through May 1996 indicates strong and continuing interest.

Clearly, the number of users does not tell the full story. Figure 7 shows the distribution of users according to the total number of requests they have made. As we would expect, the largest group of users are the ones that have simply surfed by, executed a small number of requests, and moved on. Examining 50 pages or so requires some investment of time and energy. These users are clearly evaluating the system and exploring some of its capabilities. This level of activity would also be consistent with colleagues or superiors looking at an ontology author's work. Executing 51-500 requests is sufficient to construct sample ontologies and do student level representation exercises. These users are developing a strong sense for the system and its capabilities. Finally, there are the users who have issued thousands of requests. These are our serious users, many of whom are doing substantial ontology development. The shape of this graph has remained fairly stable over time.

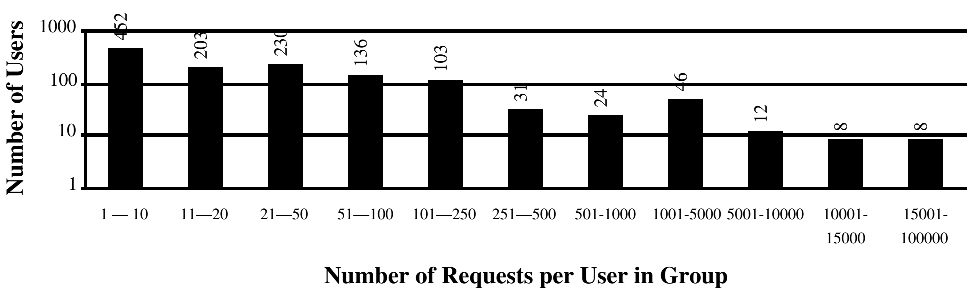

Figure 7: Users are grouped by the number of requests they make.

Figure 8 shows the distribution of requests by top-level Internet domain. This graph underestimates the amount of European use, because it does not include figures from the mirror site that was established in Madrid in November, 1995. At that point, many of the European users switched to the mirror site to benefit from improved bandwidth. Approximately 20 percent of the 734,000 requests logged came from KSL users. While the overwhelming majority of requests have been issued by U.S. educational users, there have been a substantial number issued by U.S. commercial users (COM) and active groups have been established in Europe, Great Britain, and Japan.

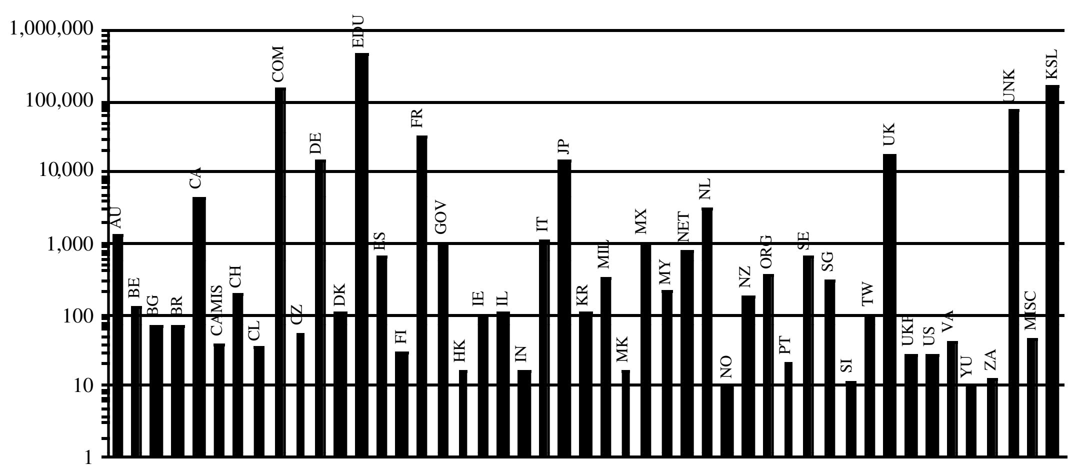

Figure 8: Number of requests by Internet domain.[4]

The usage patterns are consistent with our expectations and with those generally found in the software industry. Only a small percent of the people who try the Ontolingua Server

[4]The domain marked "Unk" accounts for all requests logged from anonymous connections. We log the requests according to the domain of the user's email address, not the current connection location.

end up using it on a regular basis. The rest try it and move on, although some of these return later when they need the capabilities of an Ontolingua Server.

Evaluating the work done with the Ontolingua Server and its impact on collaboration is difficult. It is clear, however, that users are constructing substantial ontologies. The subject matter of the user contributed ontologies is varied. The more mature ontologies include metadata for genome database integration, satellite image metadata, enterprise integration, products and product catalogs, oscilloscopes, semiconductors, robotics, solid modeling, drugs, medical terminology, the IEEE standard 1175 for tool interconnections, and many others. Some of the larger ontologies, such as the one for medical terminology, which contains over two thousand definitions, have been imported from existing projects. Others, such as the IEEE 1175 ontology, which contains over one hundred and forty definitions, have been constructed entirely through interactions with the Ontolingua Server.

In addition to ontology construction efforts, the Ontolingua Server is also being used in projects to provide run-time access to ontologies. The T-Helper medical application is an outpatient computer-based record system for patients with human immunodeficiency virus (HIV). In order to determine if patients are eligible to participate in clinical trials, it uses an ontology of drug types and specific drugs. It queries the Ontolingua Server to help determine if drug-related eligibility criteria are met (Gennari et al. 1995). The SHADE project is addressing many information system interoperability issues. The Ontolingua Server has been used to define several large ontologies for satellite image metadata (e.g., for the Federal Geospatial Metadata Standard, the Terramar satellite image database). Client-side tools are being developed to view the metadata and define mapping relations between concepts in them. These client-side tools extract the concept definitions from the ontologies stored on the Server.

The level of use that the Ontolingua Server has experienced indicates that the system is filling an important niche and that we are meeting our goal of reaching a wide audience and providing it with reliable useful tools for building and using ontologies.

5. CONCLUSION

We have described an implemented architecture for distributed collaborative ontology development and use that exploits the world wide web protocols to provide access to a growing user community.

We presented an inclusion model for ontologies that enables users to assemble new ontologies rapidly from existing ones in a repository. This model makes a clean separation between its simple formal semantics and the input/output properties of the system that uses it. The formal model handles simple inclusion, polymorphic refinement, restrictions, and circular inclusion dependencies. The input/output model yields succinct readable external representations and is transparent to users.

We described our web-based ontology editing environment (Gennari et al. 1995) that is integrated with the Ontolingua Server and that incorporates this inclusion model. In addition to providing individual users with a rich many-featured editing environment, this server also supports collaboration between distributed groups of users, and provides access to a growing repository of ontologies. The Ontolingua Server also provides a vital publishing medium for ontologies because hypertext pointers can reliably point to any document in the publicly accessible library of ontologies.

Finally, we presented empirical evidence that our approach to disseminating and providing tools to promulgate the use of ontologies is effective. Our tools and the infrastructure that supports them have been extremely reliable (with a 99.9% uptime). The infrastructure scales well and currently supports several hundred users without performance problems. Users have been able to construct substantial ontologies with the web-based editor including ontologies with many hundreds of definitions. Several of these construction efforts have been collaborative with users distributed world-wide. The server architecture also supports programmatic queries from remote software agents that interrogate the server about the definitions of terms in ontologies and possibly modify them.

Constructing ontologies is a difficult, time-consuming process. The tools that we have been developing help to amortize this effort across many users and multiply the benefits across many uses.

The Ontolingua Server is available for public use through

```
http://www-ksl-svc.stanford.edu:5915/
```

6. ACKNOWLEDGMENTS

This research was supported by a grant from ARPA and NASA Ames Research Center (NAG 2-581), NASA Ames Research Center under contract NCC2-5337, and through CommerceNet under contract CN-1094 (TRP #F33615-94-4413).

We would also like to acknowledge the valuable contributions of Wanda Pratt, Rupert Brauch, and the users of Ontolingua Server.

7. BIBLIOGRAPHY

Fikes, R., M. Cutkosky, T. Gruber, and J. van Baalen. 1991. Knowledge Sharing Technology Project Overview. KSL 91-71. Stanford University, Knowledge Systems Laboratory.

Genesereth, M. R. 1990. The Epikit Manual. Epistemics, Inc. Palo Alto, CA.

Genesereth, M. R. 1991. Knowledge Interchange Format. In Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference, ed. James A. Allen, Richard Fikes, and Erik Sandewall:599-600. Cambridge, MA: Morgan Kaufmann.

Genesereth, M. R. and R. E. Fikes. 1992. Knowledge Interchange Format, Version 3.0 Reference Manual. Logic-92-1. Computer Science Department, Stanford University.

Gennari, J. H., D. E. Oliver, W. Pratt, J. Rice, and M. A. Musen. 1995. A Web-Based Architecture for a Medical Vocabulary Server. In Nineteenth Annual Symposium on Computer Applications in Medical Care. New Orleans, LA.

Gruber, T. R. 1992. Ontolingua: A mechanism to Support Portable Ontologies. KSL 91-66. Stanford University, Knowledge Systems Laboratory.

Gruber, T. R. 1993. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition 5 (2): 199-220.

Karp, P. D., K. Myers, and T. Gruber. 1995. The Generic Frame Protocol. In 14th International Joint Conference on Artificial Intelligence. Montreal, Canada.

MacGregor, R. 1990. LOOM Users Manual. ISI/WP-22. USC/Information Sciences Institute.

Mitchell, T. M., J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette, and J. C. Schlimmer. 1989. Theo: A Framework for Self-Improving Systems: National Science Foundation, Digital Equipment Corporation.

Mowbray, T. J. and R. Zahavi. 1995. The ESSENTIAL CORBA: System Integration Using Distributed Objects.: John Wiley and Object Management Group.

Rice, J., A. Farquhar, P. Piernot, and T. Gruber. 1996. Using the Web Instead of a Window System. In Conference on Human Factors in Computing Systems (CHI96):103-110. Vancouver, CA: Addison Wesley.