Tools For Assembling Modular Ontologies in Ontolingua

Richard Fikes, Adam Farquhar, James Rice

Knowledge Systems Laboratory
Stanford University
Stanford, CA 94305
{axf, fikes, rice}@ksl.stanford.edu

Abstract

The Ontolingua ontology development environment provides a suite of ontology authoring tools and a library of modular, reusable ontologies. The environment is available as a World Wide Web service and has a substantial user community. The tools in Ontolingua are oriented toward the authoring of ontologies by assembling and extending ontologies obtained from a library. In this paper, we describe Ontolingua's formalism for combining the axioms, definitions, and words (non-logical symbols) of multiple ontologies. We also describe Ontolingua's facilities that enable renaming of words from multiple component ontologies and that provide unambiguous mapping between words and text strings during input and output. These features support cyclic inclusion graphs and enable users to extend ontologies in multiple ways such as adding simplifying assumptions and extending the domains of polymorphic operators.

Introduction

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.

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 specify ontologies in a reusable and composable form so that large portions of an ontology for a given application can be assembled from a collection of existing ontologies held in an ontology library.

We have fielded an ontology development environment called *Ontolingua* (Gruber 1992) (Farquhar et al. 1996) that provides a suite of ontology authoring tools and a library of modular, reusable ontologies. The environment is available as a World Wide Web service (`http://ontolingua.stanford.edu`) and has a substantial user community at many sites. The tools in Ontolingua are oriented toward authoring ontologies by assembling and extending ontologies obtained from Ontolingua's library. The design of the web-based interface and the underlying infrastructure is detailed in (Rice et al. 1996).

In this paper, we describe Ontolingua's formalism for combining the axioms, definitions, and words[1] of multiple ontologies. We also describe Ontolingua's facilities that enable a word to be spelled differently in multiple component ontologies and that provide unambiguous mapping between words and text strings during input and output. These features of Ontolingua support cyclic inclusion of one ontology in another (e.g., `A` included in `B` included in `C` included in `A`), which we believe to be unique to Ontolingua, and they enable users to extend included ontologies in multiple ways such as by adding simplifying assumptions or domain restrictions and by extending the domains of polymorphic operators.

The Ontolingua Representation Language

The original Ontolingua language (Gruber 1993) was designed to support the design and specification of ontologies with a clear logical semantics. To accomplish this, Gruber extended the Knowledge Interchange Format (KIF) (Genesereth and Fikes 1992). KIF is a monotonic first order logic with set theory that has a linear ASCII syntax; includes a sublanguage for defining named functions, relations, and object constants; and supports reasoning about relations, functions, and expressions by including them in the domain of discourse. Gruber extended the syntax of the KIF definition sublanguage to provide additional idioms that frequently occur in ontologies and added a *Frame Ontology* to enable ontologies to be specified in an object-oriented style using

[1] By *word*, we mean non-logical symbols — the names of relations, functions, and object constants.

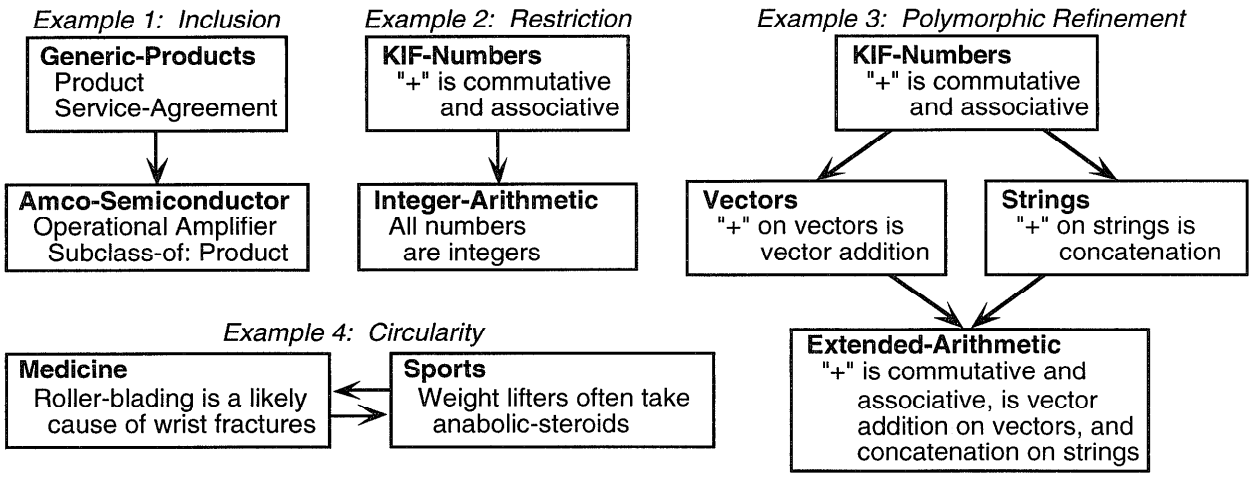

Figure 1: There are many different types of relationships between ontologies.

familiar notions such as `Class`, `Subclass-Of`, `Slot`, `Slot-Value-Type`, `Slot-Cardinality`, and `Facet`.

For the Ontolingua Server, we have extended the original language and ontology development facilities in significant ways. First of all, we have developed an object-oriented external presentation for the Ontolingua language for use by ontology developers. The *internal representation* of an ontology is always expressed as a set of KIF axioms and a set of words. That is, internally, an ontology is a first-order axiomatic logical theory. The presentation is the manner in which these underlying axioms and words are viewed and manipulated by a user. The presentation in the Ontolingua Server's browsing and editing interface is tailored for object-oriented or frame-language descriptions of a domain of discourse. The vocabulary used in this presentation is defined in the Frame Ontology.

A key property of the extended Ontolingua Language and its presentation in the Ontolingua Server is that axioms which do *not* fit into the frame formalism *are allowed*. Such axioms are displayed as augmentations to frames or as the content of relation and function definitions. Thus, the frame language presentation does not restrict the expressiveness of the ontology. This is important for a development environment for sharable ontologies, since, unlike an inference tool or a traditional knowledge representation tool for which tractability is paramount, a core objective of an ontology development environment is to support comprehensive models of a wide range of domains. For example, if a developer wishes to state the disjunction that a pass grade for an exam is equivalent to an `A`, `B`, or `C`, an ontology development environment must allow that sentence to be expressed, even though many reasoning systems may not be able to make full use of the disjunction.

Assembling Ontologies from Components

Another major set of extensions that we have made to the original Ontolingua language and system is a set of facilities that explicitly supports the composition of ontologies by assembling, extending, and refining ontologies from a library. These extensions are the primary focus of this paper.

Our objective is to make ontologies useful in a wide range of activities. 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. Ontolingua's new facilities are intended to promote that minimization and amortization by enabling ontology writers to reuse existing ontologies in flexible and powerful ways.

The original Ontolingua language provided limited support for defining ontological modules in the form of a directed acyclic graph (DAG) of named ontologies. Our users found this simple model to be inadequate in several ways, which we elaborate below. Furthermore, the module system did not have a clearly articulated semantics, which was in sharp conflict with the basic goals of the language. The current version of Ontolingua, described here, has a clear formal semantics and allows users to reuse existing ontologies from a modular structured library by *inclusion*, *polymorphic refinement*, and *restriction*.

Figure 1 shows several motivating examples that are drawn from our ontology building experience. Example 1 shows the simplest relationship between ontologies: *inclusion*. The developer of an `Amco-Semiconductor` product ontology needs to represent basic information about products, prices, services, etc. The developer does so by including the entire contents of the `Generic Product` ontology from the ontology library without modification.

In Example 2, 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. Note that multiple incompatible restrictions of an ontology are possible. For instance, an `Irrational-Arithmetic` ontology could also restrict `KIF-Numbers` by restricting all numbers to be irrationals.

In Example 3, 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, etc.). The author 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 4, 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 needs to refer to 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 words from the other.

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, and 4 to be represented, but it has three key drawbacks: First, it is impossible to restrict definitions (Example 2). Second, by eliminating modularity, it makes understanding and evaluating ontologies extremely difficult. Third, because of its flat space of axioms, it cannot handle conflicting views simultaneously. Authors using systems like this often resort to lexical conventions to discriminate between words (e.g., `+`, `vector_+`, `string_+`). Without automated support, such conventions are difficult to enforce. Furthermore, enforcing them may not even be desirable. In Example 3, 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 DAG of inclusion relations between "theories" such as provided by Genesereth's Epikit (Genesereth 1990). 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 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 (MacGregor 1990) also provides a DAG of inclusion relationships between "contexts". LOOM is a description logic system that distinguishes between concept definitions, which hold globally across all contexts, and assertions, which are partitioned by contexts. In addition to partitioning assertions, the contexts also provide a namespace mechanism so that concepts in different contexts may have the same name. LOOM relaxes the DAG constraints by allowing references to symbols across contexts. The cross context references cause minimal type checking to be performed, but do not otherwise effect the definitions. The LOOM mechanism has been driven by pragmatic considerations and developed to support hypothetical reasoning about instances. It conflates the declarative semantics, as defined by the axioms, with pragmatic information about which axioms to apply during problem solving.

In addition to knowledge representation systems, programming languages have addressed related issues with mechanisms for modules and namespaces. The Common Lisp package system, for example, shares several features with our approach — although the package system was designed for namespace management, not managing axiomatic definitions. Below, we will discuss the method used by Ontolingua to convert between input/output streams and the words in its axiomatic internal representation. This process is analogous to the way that the Lisp reader uses its package system to intern and print symbols. Similar to the knowledge representation systems, package inclusion relations are restricted to a DAG, the spelling of a symbol is globally fixed, and symbols can only be associated with a single definition of a given type.

There are two aspects to our solution: (1) an ontology inclusion operation that provides explicit support for the assembly of component ontologies, and (2) mechanisms for mapping unambiguously between text strings and words from multiple component ontologies during input and output.

Adding Ontology Inclusion to the Representation Formalism

In order to encourage 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 words (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

[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. Unfortunately, this may require the entire structure of the ontology library to be changed to add a single axiom.

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 (transitively) 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.

Ontolingua eliminates vocabulary conflicts among ontologies in its axiomatic internal representation by making the vocabulary of every ontology disjoint from the vocabulary of all other ontologies. That is, in the internal representation, the word spelled "N" in the vocabulary of ontology A is different from the word spelled "N" in ontology B. Thus, each ontology provides a local namespace for the words defined in that ontology.[3]

Given that vocabularies are disjoint, Ontolingua can assume in its axiomatic 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 union of sets of axioms.

Ontolingua allows users to state explicit inclusion relationships between ontologies and implicitly creates inclusion relationships based on words in axioms. That is, if an ontology A contains an axiom that references a word in the vocabulary of ontology B, then the system implicitly considers B to be included in A. This inclusion rule ensures that the axioms specifying the "meaning" (i.e., restricting the possible interpretations) of the referenced word are a part of the ontology in which the reference occurs. A more refined rule could include only those axioms that could affect the possible interpretations of the word, but we have not developed such a rule.

Resolving Word References in Assembled Ontologies

The inclusion operation added to the Ontolingua Server's internal representation formalism provides a powerful, simple, and unambiguous way for ontologies to be assembled and reused. However, in order to eliminate ambiguity, it requires words to be given clumsy extended unique spellings that may be unknown to the user. Moreover, it does not allow a user to perform important operations such as changing the spelling of words from included ontologies or selectively controlling which words are to be imported from included ontologies or exported to other ontologies. Ontolingua solves these problems with additional capabilities that are a part of its facilities for converting between spellings in input/output streams and words .

A word is assumed to be defined in some ontology and to have a spelling that is distinct from any other word defined in the same ontology. The ontology in which a word is defined is called that word's *home ontology.* Similarly, each ontology has a spelling that uniquely distinguishes it from any other ontology.

The mapping between text stings in an input/output stream and words is defined from the *perspective* of a given ontology. For example, if a word spelled "N" is defined in ontology A and a different word spelled "N" is defined in ontology B, then from the perspective of A, the input text "N" is interpreted as "the word spelled "N" defined in A", whereas from the perspective of B, the input text "N" is interpreted as "the word spelled "N" defined in B".

The default perspective from which any given text string is to be interpreted is established unambiguously by the Ontolingua Editor or the ontology source file, but it can be explicitly specified by attaching a suffix consisting of the character "@" followed by the name of an ontology. So, for example, the word N whose home ontology is A can be unambiguously and globally spelled "N@A". A string that includes the @«ontology name» suffix is said to be a *fully qualified reference.* Fully qualified references enable words defined in any ontology to be referenced in any other ontology.

The @«ontology name» suffix can be omitted from word references in an Ontolingua input stream and is omitted from word references produced by Ontolingua in its output streams when the word's spelling itself is unambiguous from the intended perspective. Such a word is said to be *recognized by spelling* from the perspective. A word is always recognized by spelling from the perspective of its home ontology (i.e., the ontology in which it is defined). Determining when a word can be recognized by spelling in an ontology other than its home ontology requires additional machinery, which we will now describe.

The Ontolingua Server's input/output facility provides a mechanism that allows a user to assign a spelling to a word which is local to the perspective of a given ontology. The word is then recognized by spelling from the perspective of the ontology in which it is renamed. A renaming is specified by a rule that includes an ontology name, a word , and a spelling that is to be used as the local spelling of the given word from the perspective of the given ontology. Given such a renaming rule, Ontolingua will recognize the local spelling as a reference to the given word when processing input in the given perspective, and will use the local spelling to refer to the given word when producing output from that perspective. So, for example, a renaming rule might specify that in ontology A, the local spelling for auto@vehicles is to be "car". This facility enables an ontology developer to refer to words from other ontologies using names that are appropriate to a given ontology and to

[3] The names of ontologies, themselves, are considered to be global and are not part of the ontological vocabulary.

specify how naming conflicts among words from multiple ontologies are to be resolved.

A word is said to be recognizable by spelling in an ontology if the spelling unambiguously identifies a word from the perspective of that ontology. In order for the test for ambiguity to be well defined, the space of words to be considered by the test must be specified. If that space is too large (e.g., all the words in all the ontologies in the Ontolingua library), then names will rarely be unambiguous. Thus, what we need are mechanisms for specifying a restricted space that is appropriate for ontologies assembled from component ontologies. We provide three such mechanisms as follows.

The first mechanism for restricting the words considered during recognition enables a word to be designated as *private* to its home ontology and therefore not renameable nor recognizable by spelling in any other ontology. The Ontolingua Server provides user commands for designating words as being public or private. However, the system considers words to be public by default so that users can ignore the public/private distinction until they encounter or want to define private words.[4]

The second mechanism for restricting the words considered during recognition associates with each ontology a set of names that are blocked in the ontology from being recognized as words from another ontology. Such names are said to *shadow* (hide) words from other ontologies. The Ontolingua Server provides user commands for editing the set of shadowing names associated with each ontology.

The third mechanism for restricting the words considered during recognition associates with each ontology a set of ontologies which are sources of words that can be recognized by spelling in that ontology. Commands are available in the Ontolingua Server for editing the set of such source ontologies, and by default Ontolingua automatically adds to this set any ontology which is explicitly included. Thus, by default, all (transitively) included ontologies are sources of words to be recognized by spelling.

The algorithm for recognizing a word from a string `"N"` in a given ontology `A` checks whether there is a word defined in `A` spelled `"N"`, then checks whether there is a word renamed to `"N"` in `A`, and then, if `"N"` is not shadowed in `A`, recursively attempts to recognize `"N"` as a public word in an ontology which is a source of recognized words for `A`.

Ontolingua uses the shadowing mechanism to prevent ambiguities from occurring in references to words defined in other ontologies by automatically blocking any spelling which would be ambiguous in the ontology. In particular, given ontologies `A` and `B`, and a situation in which there is a word that is recognized by spelling `"N"` in `A`, the system automatically shadows `N` in `A` when:

- A new word is defined that would be recognized by spelling `"N"` in `A`;
- A word is renamed to `"N"` in `A`;
- There is a word which is recognized by spelling `N` in `B` which is different from the word that is recognized by spelling `"N"` in `A`, and `B` becomes a source of recognized words for `A`.

Note that although the recognition of words by spelling is independent of the order of operations that occurred before a string is read from an input stream (or entered into an output stream), it does not change the interpretation of names read before an operation is done. That is, spellings in input streams are interpreted with respect to the state of the system at the time the stream is read.

Discussion

To summarize, consider how Ontolingua supports ontology inclusion, circular dependencies, and polymorphic refinement by reconsidering the examples from Figure 1.

The developer of the `Amco-Semiconductor` products ontology would explicitly establish the ontology inclusion relationship in Example 1 either as part of the definition of that ontology or as an editing operation after the ontology has been defined. When the `Generic-Products` ontology is included in the `Amco-Semiconductor` products ontology, the system automatically adds the `Generic-Products` ontology to the Amco ontology's set of sources of recognized names. As a result, public words from the `Generic-Products` ontology, such as `Service-Agreement`, whose spellings do not conflict with other spellings from the perspective of the Amco ontology would then be recognized in the Amco ontology and therefore could be referred to from the perspective of that ontology by spelling (e.g., `Service-Agreement`) without the `@Generic-Products` suffix.

Examples 2 and 3 illustrate the restriction in one case and the extension in another case of a function (i.e., "+") defined in an included ontology. In example 2, the author of the `Integer-Arithmetic` ontology restricts numbers to be integers in that ontology by augmenting the definition of class `Number` in the `Integer-Arithmetic` ontology so that it is a subclass of `Integer`. That augmentation results in the addition of the following axiom to the `Integer-Arithmetic` ontology:

```
(=> (Number ?x) (Integer ?x))
```

The `+` function defined in ontology `KIF-Numbers` is then restricted to apply only to integers in the `Integer-Arithmetic` ontology.

In example 3, the function `+` is extended to become a polymorphic operator. The author of the `Vectors` ontology explicitly includes `KIF-Numbers` in `Vectors` and augments the definition of `+` with the following axiom to make `+` equivalent to `VectorAdd` when the arguments are vectors:

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

```
(=> (and (Vector ?x) (Vector ?y))
    (= (+ ?x ?y) (VectorAdd ?x ?y)))
```

Note that since KIF-Numbers is explicitly included in Vectors, the spelling "+" in this axiom is recognized as referring to the word + defined in KIF-Numbers. Similarly, the author of the Strings ontology explicitly includes KIF-Numbers in Strings and augments the definition of + with the following axiom to make + equivalent to concatenation when the arguments are strings:

```
(=> (and (string ?x) (string ?y))
    (= (+ ?x ?y) (Concat ?x ?y)))
```

As before, since KIF-Numbers is explicitly included in Strings, the spelling "+" in this axiom is recognized as referring to the word spelled "+" defined in KIF-Numbers. When the author of Extended-Arithmetic includes both Vectors and Strings in Extended-Arithmetic, + in that ontology polymorphically applies to numbers, vectors, and strings.

The polymorphic refinement of + in Example 3 is a case in which some of the subtle properties of implicit ontology inclusion become apparent. If the Extended-Arithmetic ontology does not explicitly include the Vectors and Strings ontologies, then references to +@Vectors and +@Strings in Extended-Arithmetic will cause KIF-Numbers to be implicitly included in Extended-Arithmetic, but will not cause Vectors or Strings to be included since both +@Vectors and +@Strings refer to a word whose home ontology is KIF-Numbers. If the axiomatic definition of the + operator is intended to apply to vectors, strings, and numbers in the Extended-Arithmetic ontology, then the author of that ontology is expected to include the Vectors and Strings ontologies explicitly (and optionally the KIF-Numbers ontology).

Finally, the circular dependencies in the Medicine and Sports ontologies of Example 4 could be established and presented by not explicitly including these ontologies in each other, and using fully qualified names to refer to words in the other ontology. For example, in the Medicine ontology, roller-blading would be entered as "roller-blading@Sports", and in the Sports ontology, steroid-tests would be entered as "steroid-tests@Medicine". The reference to roller-blading in the Medicine ontology will cause the axioms of the home ontology of the word roller-blading@Sports to be implicitly included in the Medicine ontology, but will not cause text strings input from the perspective of the Medicine ontology to be misinterpreted as words from the Sports ontology. The public words from the Sports ontology will not be recognized in the Medicine ontology because the inclusion is implicit; the Sports ontology does not become a source of recognized names for the Medicine ontology.

Conclusions

We have described mechanisms in the Ontolingua ontology development environment that support authoring of ontologies by assembling and extending reusable ontologies obtained from an on-line library. We described a formalism for combining the axioms, definitions, and words of multiple ontologies. We also described Ontolingua's facilities that enable renaming of words from multiple component ontologies and that provide unambiguous mapping between words and text strings during input and output. These features of Ontolingua support cyclic inclusion graphs and enable users to extend ontologies in multiple ways such as adding simplifying assumptions and extending the domains of polymorphic operators. They include default settings that provide intuitive system behavior in most situations without any effort on the part of the user, while providing detailed controls when needed by the sophisticated user.

Acknowledgments

This research was supported by the Defense Advanced Research Projects Agency under contract N66001-96-C-8622. We would also like to acknowledge the valuable contributions of Sasa Buvac, Angela Dappert, Robert Engelmore, Wanda Pratt, and the users of Ontolingua.

References

Farquhar, A., Fikes, R., & Rice, J. 1996. The Ontolingua Server: a Tool for Collaborative Ontology Construction. Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop. Banff, Canada. November 9-14.

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

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

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

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.

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.