

Semi-automatic Construction of a General Purpose Ontology

Andrew G. Philpot

Michael Fleischman

Eduard H. Hovy

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
+1 (310) 822-1511
{philpot, fleisch, hovy@isi.edu}

ABSTRACT

Ontologies are hierarchically organized networks of conceptual information. Used to systematize and model domain knowledge, they play an important role in artificial intelligence, natural language processing, information integration, electronic commerce, and related fields. In this paper, we briefly describe the characteristics of ontologies, and then discuss the Omega ontology, a large general-purpose ontology created semi-automatically from WordNet, Mikrokosmos, and a newly created upper semantic model, and note the relationship between Omega and its predecessor Sensus. We detail the procedures used to create the Omega base ontology and to merge in new ontological information, such as custom ontologies derived from glossaries and instances obtained by data mining. We survey some of the applications in which we have applied Omega. We highlight the tools used and characteristics of the Lisp environment which facilitate the development and maintenance of the ontological structure. We conclude by discussing current and future work.

Categories and Subject Descriptors

I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods – representation languages, semantic networks, frames and scripts; I.2.7 [Artificial Intelligence]: Natural Language Processing – language parsing and understanding, text analysis.

General Terms

Experimentation, Languages.

Keywords

Ontologies, artificial intelligence, natural language processing, question answering, information integration.

1. INTRODUCTION

Ever since Aristotle, it has been remarked that the human mind tends to organize, differentiate, and classify information. Hence hierarchically structured networks of information are very common in information processing, as well as in everyday human

life. Artifacts as mundane as telephone numbering schemes, as comprehensive as the Linnean biological nomenclature system, and as ambitious as CYC [8] can all be considered representations of ontological knowledge. Ontological representations based on subtype relationships provide a basis for AI reasoning and representation for a variety of activities. Ontologies may be embedded within sophisticated reasoning mechanisms or they may be simple taxonomies whose power is limited to their ability to discriminate between objects and their subtypes.

In this paper, we describe the Omega ontology, a Lisp-based composite constructed from various other hierarchically arranged information sources, detailing its parentage, construction, applications, and unique affinity to the Lisp environment. We close with discussion of future work.

2. HISTORY

The Omega ontology is the intellectual descendant of Sensus [7]. Sensus, developed at USC/ISI during the 1990's, was one of the first large ontologies. Its primary use was in natural language processing; the Sensus Ontosaurus browser was one of the first web-based applications developed for exploring ontological information. Sensus was originally developed by amalgamating WordNet [2] from Princeton with ISI's Penman Upper Model [1], using the Sensor knowledge representation (or "frame") language. It was used to support machine translation, as the central cross-language organizing structure for lexicons of Spanish, Japanese, and English. However, the general nature of Sensus provided more general utility, as became apparent for other application domains. Several years ago, the authors reconstituted Sensus using the then-current version of WordNet (1.6) and applied it to support single-point information access across multiple databases. The lessons learned during this reimplementation and the applications fielded with Sensus2 led us to undertake the construction of a new ontology, firmly within the tradition of Sensus, but addressing several new concerns:

- Given the wider variety of envisioned uses for the ontology, a more domain-independent upper structure model was desired, as compared with the cognitive psychology basis of WordNet and the linguistics focus of Penman.
- In order to support more and varied modes of ontology construction and manipulation, a cleaner delineation

between the lexical items (words) and concept terms needed to be made; similarly, the separation between the Sensor KR layer and the Sensus ontology proper needed to be improved. In particular, support for multiple co-existing ontological spaces, and *ad hoc* newly defined relations were needed.

- The Ontosaurus browser, based on *c.* 1995 web technology, had become difficult to maintain and enhance. Besides in-core processing and ontology browsing, we hoped to use the ontology to provide other services. It was thus determined that a new browser and application core could be more closely integrated into the Lisp ontology base.

3. ONTOLOGY CONSTRUCTION

3.1 Basis

The original Sensus was largely composed of content from WordNet 1.4, articulated into a natural-language focused structure by use of the Penman Upper Model [1]. The value of WordNet (110,000 concepts in version 1.7.1) was always apparent and would remain the core contributing source. While Penman's upper structure at times made certain forms of reasoning less facile than desired, it was in the intermediate layer where the need for better structure was most apparent. Accordingly, we decided to use the 6000-node Mikrokosmos model [9], obtained via license, hoping to leverage its well developed middle structure. However, we also wanted a domain-neutral upper model. The result of this three-way merger is the Omega ontology. As in the case of Sensus, Omega was much too large for any person or team to construct manually. We therefore employed a semi-automated regimen to entwine together the various components.

3.2 Upper Structure

At the top, a new upper model (NUM) was generated by hand in order to form a common foundation for the lower models of the initial two source ontologies (Sensus/WordNet1.7 and Mikrokosmos). The NUM is an entirely novel hierarchy created to maximize coverage over the lower source ontologies. In this way, design decisions (such as which concepts to include and what partitions to prefer) were made in order to insure that all concepts in the lower models of the source ontologies would fit under the NUM. Thus, the NUM differs from other upper ontologies whose organization is motivated by such concerns as language (Penman Upper Model), translation (Mikrokosmos), and cognition (WordNet).

The NUM begins with a single most-general concept (|Summum Genus|¹) and terminates with approximately 200 leaf concepts such as |Religious-Event| and |Linguistic-Object|. Local lattice points, capturing in a single node the confluence of several independent differentiating features to express sets of mutually exclusive alternatives, were included where appropriate, to maximize coverage and efficiency. The entire NUM is made up of approximately 400 concepts, with subtree depths ranging from 3 to 7, and an average branching factor of

¹ In this paper, we use identifiers enclosed in vertical bars and sometimes employing ontology vocabulary prefixes to denote concepts: e.g., |Animal|, O@::|Intangible-Object|.

approximately 3. An example of the lattice articulation is shown in Figure 1 below.

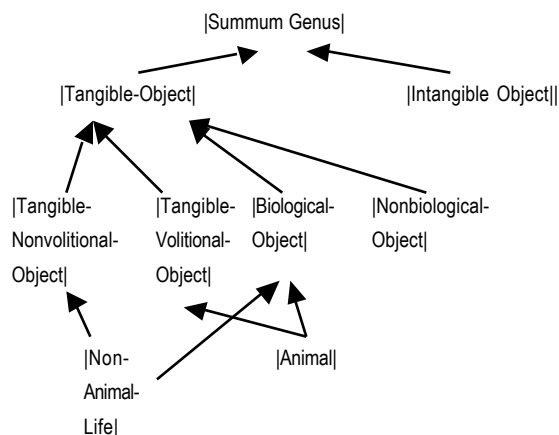


Figure 1. Lattice Structure of New Upper Model.

denotes a mutual exclusion pair.

3.3 Middle and Lower Structure

The middle and lower component lower models used in constructing Omega were Sensus/WordNet 1.7.1 and MIKROKOSMOS. By Sensus/WordNet 1.7.1, we indicate an extraction from WordNet 1.7.1 into Sensus format (the original Sensus used WordNet 1.4, while Sensus II used 1.6). We briefly describe these two components, the native structure and organization and the approach we take to re-express them in our ontology framework.

3.3.1 WordNet

WordNet gathers words with similar meanings into synonym sets or "synsets". Each synset has associated with it pointers to the lexical items that populate it, hyponyms and hypernyms, and other properties such as part, substance and member holonyms, pertainyms, gloss, etc. Adjectives are further grouped into adjective satellite sets; the center of each set (e.g., "dry") contains the general characteristic, while the satellites ("arid", "rainless", "thirsty", etc.). We re-expressed each synset as a concept in Sensor (the knowledge base language used in Sensus); the satellite-to-center pointer (for adjectives, when available) or the hypernym pointer (for other parts of speech) was reformulated as Sensor's DIRECT-SUPERCLASS relation. Similarly, the part, substance and member holonym pointers and pertainym pointers became Sensor's DIRECT-PART-OF, DIRECT-SUBSTANCE-OF, DIRECT-MEMBER-OF, and PERTAINS-TO. (We did not need to track the hyponyms since Sensor has inverse relation maintenance). The gloss was deconstructed using heuristics into definition, usage, and examples, and these were attached to the constructed concept as well. Internal WordNet attributes such as sense key and index offset were retained. Each concept was associated to lexical item ("word") instances generated from the attached WordNet words. Finally, the concepts were globally renamed using a procedure, detailed in section 5.2, which maintains uniqueness, provides disambiguation, and improves perspicuity.

3.3.2 Mikrokosmos

The native format of Mikrokosmos was already quite similar to that of Sensor. The primary transformation needed was to map the ternary relation model (object, relation, aspect) of Mikrokosmos into the binary (object, relation) representation one used in Sensor. For each relation deemed similar to a Sensor relation, the appropriate aspect (generally SEM or VALUE) was chosen and mapped directly (e.g., Mikrokosmos relation/ISA aspect SEM became DIRECT-SUPERCONCEPT); the INV aspect was represented using Sensor's inverse for the relation, if it existed. Other unmatched aspects, and relations not similar to those in sensor were flattened (e.g., relation EMPLOYER-OF/aspect SEM became a newly created Sensor relation EMPLOYER-OF--SEM) and retained verbatim.

3.4 Linking Upper and Lower Models

With the NUM in place, the upper models of the source ontologies were removed and the remaining concepts linked into the leaf nodes of the NUM. Since we had reconstructed a Sensus image of WordNet *de novo* from WordNet 1.7.1, there was no Penman upper model, and only a few concept nodes deriving from WordNet proper needed to be removed. For Mikrokosmos, we considered the top 4 levels to be the upper model in general (and removed them), although alternative depths were chosen when intuition so indicated.

3.4.1 Initial Attachment

We then manually assigned each Sensus and Mikrokosmos node to one or more of the upper structure nodes, either as identical to an upper structure concept node (via a "merge" macro), or as a subconcept ("move"), in each case bringing along the entire subconcept structure. The result of this process was one single top region (i.e., the NUM) below which hang strands of concepts once linked into the upper ontologies of Mikrokosmos and Sensus. These strands were linked together at the leaf nodes of the NUM and form two "curtains" that hang below. On one curtain exists a strand of Mikrokosmos concepts and on the other side exists a strand of Sensus concepts (see Figure 2).

3.4.2 Bottom-up Cross-Ontology Merging

In the next phase we "sewed up" the curtain by first merging the leaf of one side of the curtain into the other, forming a "concept bubble." Then, the bubble was flattened out by merging the interior elements of one side of the curtain into the other.

One the strands of source concepts are linked into the NUM, the leaf concepts of these strands must be tied into each other to form the concept bubbles. This was accomplished by comparing each leaf concept in each of the Mikrokosmos strands to all of the concepts in the corresponding Sensus strand and using a machine learned classifier to suggest which pairs are suitable for merging. Once the classifier had made its suggestions, a human annotator went through and decided which, if any, Sensus concept should merge with the Mikrokosmos leaf. The learned classifier was trained on pairs of concepts from older versions of Sensus and Mikrokosmos that had been hand determined to be either appropriate or inappropriate for merging [3].

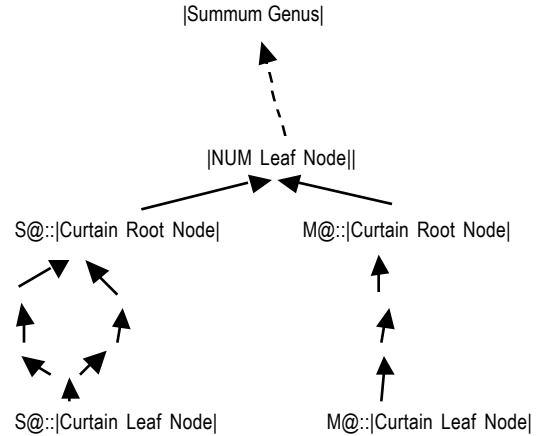


Figure 2. Merging Ontology "Curtains".

M@ indicates concepts derived from Mikrokosmos. **S@** indicates concepts derived from Sensus/WordNet 1.7.1. Concept nodes interior to the curtains have been elided.

These pairs were generated from an earlier system that filtered out obviously inappropriate concepts based on name and definition similarity, leaving only similar pairs to be hand annotated. We enhanced this system by generating features for each pair based upon a number of different matching metrics which examine both the names of the concepts and their definitions. Because only pairs of similar concepts were available for training, initial results were poor on real world cases in which the vast majority of concept pairs are very dissimilar. In order to handle this, we trained another decision rule classifier to distinguish between pairs of concepts that are similar, though not necessarily suitable for merging ("near mismatch"), and very dissimilar. For the similar cases we used both the positive and negative examples from [3] and for the dissimilar cases we used randomly selected pairs of concepts. Using the same features as above, this decision rule classifier achieves an improved accuracy on a 10-fold cross validation.

Because of the time required to calculate features for each pair and the vast number of possible pairwise comparisons between Sensus and Mikrokosmos, the model could not be exhaustively employed over all concept pairs. Instead, each Mikrokosmos leaf node for each strand was compared against only those Sensus concepts that appeared in the corresponding Sensus strand. This heuristic search limitation is based upon the intuition that any leaf Mikrokosmos concept, if it is to merge with any Sensus concept at all, must merge with one that shares the same upper ontology concept. In practice, however, this is not the case. Often, concepts under different upper ontology frontiers are too similar to be seen as distinct. For example, because of the different perspectives sometimes taken by the source ontology builders, the concept |library| in Sensus is classified under |building|, while it is classified as an |organization| in Mikrokosmos. Clearly, both perspectives are valid, but equally clearly, they lead to contradictory implications if both are accepted: an organization can make decisions (like buy books, hire people, etc.) while a building cannot. Treating this kind of multi-

perspective fluidity has been a perennial problem in Knowledge Representation. These extra-strand concepts, however, are never examined by the classifier and thus, never presented to the human annotator. For this reason, we included along with the classifiers' suggestions for merging, a list of all Sensus concepts with (word) names that are identical to the Mikrokosmos leaf concept. Once the classifier's suggestions are calculated for a given Mikrokosmos leaf concept, the five proposed merge concepts with highest confidence are presented to a human annotator (along with the Sensus concepts having identical names) in a web-based CGI interface developed especially for this purpose. The annotator then determined which, if any, of the concepts should be merged with the Mikrokosmos leaf.

3.4.3 Multiple Perspectives and the "Shishkebab" Approach

In order to handle the multi-perspective problem, we introduced a new construct we call a shishkebab. A shishkebab is a relation that holds with enough regularity between ontology nodes that refer to the same concept as seen from different perspectives: for example, libraries, schools, hospitals, museums, etc., viewed as building or as organization,. Here, identity is not associated with single nodes, but rather with sets of nodes, each reflecting a different aspect of the individual concept. These nodes are "skewered together" to make a unique whole, as if on a shishkebab. We define the concept itself to be the whole shishkebab, the sum of its individual senses, each of which may have a different superconcept.

Shishkebabs are particularly useful for ontology merging as they allow a different level of concept identification to accommodate the differing perspectives of ontologists. Initially, we have found around 400 patterns occurring frequently enough to be called shishkebabs. Further investigation of shishkebab patterns within the growing ontology are part of our ongoing research.

3.4.4 Flattening Remaining Bubbles

Having tied each leaf of the Mikrokosmos concept strand to some Sensus concept, the final phase of ontology merging consists of flattening the concept bubbles that remain.

The flattening of concept bubbles is essentially an extended version of the concept tying described in the previous section. The task is, given two arrays of concepts whose first and last elements are merged, run the decision rule classifier over the Cartesian product of all intervening concepts. We can reduce the number of proposals, however, by looking not at individual pairs of concepts, but at all possible orderings of these proposals. By pruning out cases of crossing alignments, we reduced the search space and presented to the linker only those merges that did not create inconsistencies. In many cases, there was still no suitable merge to flatten out the concept bubble. This was often due to differences in ontological precision, *i.e.*, one ontology contains more fine-grained conceptualizations than the other. However, in some cases there are no suitable merges because the decisions made to partition the concepts were incompatible between ontologies. The existence of such cases is particularly interesting because it shows that even in such constrained circumstances, ontologists still manage to partition their worlds differently.

4. APPLICATIONS

Omega and its predecessor Sensus have been used in several different applications including machine translation, information integration, and question answering. Sensus was used in machine translation and other natural language processing.

A later version of Sensus (Sensus II based on WordNet 1.6) formed the central integration structure for the Energy Data Collection information integration project [4]. A special subontology was created to model the concepts used in energy price and volume time series. The concepts in this ontology were linked one-to-one to concept definitions in a description logic, articulated over properties such as area of measurement, frequency, seasonal adjustment, and associated product (*e.g.*, grade and formulation). Expert users could supply many or all defining properties; naive users could supply fewer ones and/or navigate incrementally through the more populated portions of the (sparse) time series space. Eventually, one or more series would be fully defined in this process, concluding in the user's access to data, obtained by invoking the SIMS query planner to dynamically create and execute an information access plan specific to the source or sources needed to respond to the query.

Another application related to energy time series was the AskCal question answering system [10]. An ATN was developed based on a general set of question types determined via a user survey to be useful in the domain. The parser used domain-specific concepts and terminal symbols drawn from a specialized subontology both during parsing (*e.g.*, areas and dates may follow the preposition "IN", but not prices) and when scoring candidate valid parses. As before, users could incrementally refine the concept definition; moreover, users could also incrementally refine the search by editing the input natural language expression or its menu-based paraphrase.

The DINO (Dino Is Not Ontosaurus) ontology browser has been developed to provide informal WWW-based access to the ontology components and to serve as a basic portal and integration platform for building project-specific interfaces such as AskCal above. Where the Sensus Ontosaurus used custom-created infrastructure involving Apache, Perl/CGI, frame status cookies, and sockets to communicate with Lisp, DINO is a straightforward Lisp-embedded WWW application, providing users with the ability to both browse the existing ontology and upload new components to it.

Omega provides many general purpose concepts, but for some applications, specialized ontologies may be needed. One area of recent work has been in automatically capturing information expressed in text corpora such as glossaries, relating it to lower-level Omega concepts, and materializing the relationships into a domain-specific child ontology [5,6]. In particular, we used Google-obtained glossaries and a custom glossary web crawler to identify likely web pages containing glossaries; then applied landmark-based and parsing techniques to identify likely supertype/subtype relationships expressed; when parent concepts consistent with the other cues from the sentence (*e.g.*, part of speech) were available, the resulting concept could be added in directly as a subclass of the found Omega concept. Applied in energy, environmental regulation, and other domains, the results were a general ontology accessible to all users, containing the ability to drill down into domain-specific application areas.

Such ontologies are useful in support of information integration and question answering.

We have also developed a socket-based command-line interface to Omega called DINOCMD. DINOCMD exposes a simple relational view of certain of Omega's key relations; for example, a user may ask for all parents, descendants, or lexical items of a given term. This mechanism is useful for a client which wants to use the Omega base for a restricted purpose and may not desire full integration with Lisp.

Access to Dino and other research prototype systems related to the Omega research effort may be obtained at <http://omega.isi.edu/>.

5. OMEGA AND LISP

Omega benefits greatly from implementation in Lisp. In this section we discuss some of the advantages the Lisp implementation gives and detail some of the implementation techniques employed. We develop using Allegro CL 6.2, and have deployed prior versions of Omega using CMUCL.

5.1 Concept and Lexical Item Representation

The fundamental elements of Omega are concepts and lexical items, or words. Both words and concepts are implemented as symbols in Lisp, and are placed into specially designated packages. The requirements of ontology concepts and the capabilities of symbols match very well. Symbols are unique within a package, they can have arbitrary properties attached to them, and they print easily. They easily support implicit, partial, and/or forward references (*e.g.*, you can check whether a concept exists, define a word in terms of the concept, then define the concept, with no special exertion required). Symbol names containing spaces or lower case letters are printed by default in the standard Lisp environment using enclosing vertical bar (`|`) syntax², and all symbols may be legally printed and read in this syntax. Since use of spaces and lower case is typical for concept and word names, we use `|` in general for notating concept and word symbols. Symbols can take on values and function definitions; we have considered making all ontology symbols self-evaluating, although that is not part of our current practice. Standard introspection tools like APROPOS and DESCRIBE are very helpful for informal browsing of the concept and lexical item spaces. Experimentally, in part because of characteristics listed above, symbols seem more "immediate" than alternative data structures (class instances, structures, database records, string identifiers within hash-tables).

The Sensor frame system is used to implement the Omega ontology. Sensor, originally developed for Sensus, internally employs property lists to record the concept-concept and concept-word relationships that make up the ontology network. Property list access is $O(n)$, yet with an average number of properties around 20, we have not experienced any noticeable performance hit due to this. It should be noted that one could postprocess the property lists after ontology creation to move more commonly referenced properties to the

² The standard also permits single-character escaping within such symbols, of course, but we have rarely encountered such in practice. Our pretty-printer always uses the vertical bar convention, in any case.

front to improve overall performance without abandoning the property list representation; we have not done this optimization.

5.2 Concept Naming

The above discussion about symbols, particularly the argument from intuition about symbol ease of use, presupposes to some degree that a symbol will have a name that is both related to the concept being modeled, easily distinguishable from other related and unrelated concepts, and of course, unique. However, in an ontology, there may be two distinct concepts for which the most natural identifiers are identical. By natural identifier, we mean the most common lexical item in use to refer to a concept. Consider the concepts:

- a) "sloping land next to river"
- b) "small container for coins"
- c) "financial institution accepting deposits"
- d) "to conduct banking transactions with"
- e) "slope in the turn of a road or track"

All would be named most naturally by the identifier "bank". Of course, in order to provide for unambiguous reference to each concept, some sort of naming and/or disambiguation mechanism may be needed. Differing approaches for resolving these name contention issues in ontology construction might include:

- **Numerically generated name.** Concept names are essentially arbitrary, so the simplest approach is to just assign arbitrary globally unique identifiers to each defined concept. The primary advantage of this scheme is its simplicity. The approach leaves a lot to be desired from the standpoint of users (browsers, later editors); even if the generated name is based on "bank" above and/or the concept is attached to a lexical item "bank", the user has a significant burden when contextualizing a concept.
- **Lattice Address.** A second way to disambiguate concepts is to generate unique identifiers based on their subclass access path or "address" within the lattice. For example, one can name the root node "1", its children "1.1", "1.2", etc. Advantages of this mechanism are relative simplicity, while retaining some contextual information: one knows that one parent of the node labeled "3.4.5" is the one labeled "3.4". Disadvantages are that in the presence of a lattice structure, more than one unique name might apply to a given node; indeed, in the presence of significant multi-parent nodes, the number of candidate names will be exponential in the ontology size. One can avoid this explosion as well, at the price on having less complete information about parent/children relationships. Moreover, while given an address, one can easily find the address of a parent concept, the meaning of any given name in isolation is hardly obvious.
- **All Associated Words.** WordNet is organized by grouping all words whose meanings are (roughly) synonymous into a synonym set, or "synset." For example, description labeled c) above in WordNet is described by the synset "depository financial institution, bank, banking concern, banking company"; while e) above is "bank, camber, cant." The advantages of this

approach are that the shared concept meaning is often very clear. Synsets include *all* lexical items known to WordNet to be accurate representations of the shared semantics, generally ordered most common first. Therein lies one of the disadvantages of this mechanism: it can often result in names which are awkwardly and (with respect to ambiguity) unnecessarily long. Paradoxically, a second disadvantage is that often they are too short: WordNet has no word other than "bank" for a) and d) above, and hence one must fall back on the synset offset (integer file position) to uniquely identify senses.

- **Parent concept reference.** The discussion above about naming concepts by virtue of their "lattice address" reflects the intuition that a concept can often be best understood by reference to its parent (or parent class). Ontologically organized systems such as CYC and TAP [12] use this principle when naming entities. The CYC base model is constructed manually; the reference manual [11] suggests disambiguating various concepts associated with "bow" using names like |Bow-BoatPart|, |BowTheWeapon|, and |Bowling-BodyMovement|. In each case, the second name constituent refers to a supertype (or super-assembly) of the sense of "Bow" in question. Similarly, the TAP KB RDF [12] labels generally are constructed using the class to which the entity belongs: so one has |Politician_Paul_Simon| and |MusicianSimon,_Paul|. The advantages of this technique are that the generated names are very clear. The chief disadvantage is that given a large enough ontology, even this technique is not guaranteed to generate unique names. In other words, a concept could have two distinct subconcepts, each of which possesses the same natural name. For example, WordNet has two different synsets that include 'prison camp', both of which are subconcepts of 'camp', namely:

- prison camp, internment camp, prisoner of war camp, POW camp -- (for political prisoners or prisoners of war)
- work camp, prison camp, prison farm -- (for trustworthy prisoners employed in government projects).

Cases such as this are common especially with regard to cases of verb intensification, imitation, or informal vs. strict uses. When naming by reference is unique, especially for nouns and some verbs, it is often effective. However, for more qualities and many verbs, the best way to disambiguate concepts might not be via the parent concept. Consider the adjectival concept 'Georgian': senses include

- "having to do with King George of England"
- "having to do with the US State Georgia"
- "having to do with the republic of Georgia, its people and/or its language"

For each of these, the best disambiguator is to which sense of 'George' or 'Georgia' the word pertains, not the parent concept.

- **Omega's hybrid approach.** In OMEGA we use a naming algorithm inspired by the one from Sensus (due to Jonathan Graehl). A set of candidate names, each telegraphing one aspect of the concept definition, is

associated with each provisional concept. Candidate name generation methods include:

- *Direct:* choose one or more of the synset words |rattlesnake|, |port,left|
- *Subclass:* distinguish with reference to immediate parent: |olive<fruit|
- *Superclass:* name with reference to a subconcept: |mob>Mafia|
- *Ancestor:* define in terms of an indirect ancestor: |barley<<grass|
- *Definition:* use definition, possibly shortened: |defacto==in reality or fact|
- *Pertains to:* attributive to attributed object: |Danish~Denmark|
- *Usage:* name using extracted usage attribute (heuristically obtained from gloss or subject domain): |tonic(music)|
- *Example:* Follow name with example usage: |carry:The senator carried his home state|

Each generated name is assigned a cost, computed as a function of its generation method and syntactic cues (*e.g.*, brevity, number of internal constituents). After generation, the generated names are assigned to concepts by choosing the name with local minimum cost, preferring globally unambiguous names, but using possibly ambiguous names when necessary. A planned future enhancement is to employ a general constraint satisfaction relaxation procedure to assign names, which should improve overall assignment quality. The procedure tends to provide reasonable and understandable names, although far in quality from the ones a human might use following the same basic methodology.

5.3 Subontologies and the Ontology Vocabulary

The key organizing data structure in Omega is the ontology vocabulary or ovocab. The ovocab corresponds to a subontology, with its own namespaces for concepts and for each language in which lexical items exist for that subontology. Most relation links involving ovocab concepts are intra-ovocab, but links to other ontologies, in particular DIRECT-SUPERCLASS links to a more general ontology, are not uncommon.

The implementation of the ovocab is as confederation of packages, one for the concept space and one for each language containing any lexical items. At present, we use ISO639 two-letter language codes by default to identify languages, but any unique language identifier could be used. As well as being placed in a dedicated registry, the packages use a naming convention to assist the reader in associated the symbols with the parent ovocab.

For example, the OMEGA base ovocab places its concepts in the O@ package, its English language lexical items into the O@EN package, and its Spanish language lexical items into the O@ES package.

By default, the all constituent packages do not use (in sense of Common Lisp's USE-PACKAGE) any other lisp packages and import only two defining macros for ease of defining new

forms. This means that any arbitrary symbol from the :COMMON-LISP or any other resident package will never conflict with a proposed concept or lexical item name. We have implemented ovocab-aware reader macros which allow for simplified reference between the concept and language namespaces of the current vocab (although use of full package syntax is more commonly done). We can use Lisp package symbol exporting to share concepts or words common to two ontologies. (Sensor implements an ALIAS relation appropriate for cases where the two codesignating concepts do not share the same name. Neither concept exporting nor aliasing is very common in our current practice.) Since ovocabs are package-based, we can programmatically add, delete, and rename them more easily than with a scheme encoding the disambiguating prefix into the name proper. Since concepts and words are symbol-based, we have both an open interface, where one can use Lisp tools such as INSPECT and APROPOS to browse the implementation of the concept base and an implemented API of ovocab-specific constructors and iterators to process the vocabulary, more suitable for programmatic processing. (We have considered using Allegro CL's package-locking mechanism to trap attempts to intern new concepts outside the scope of the API; in particular, to avoid polluting the concept space due to an interactive typo, but have not implemented this).

The net result of all these characteristics is a mechanism for simple and natural access to ontology objects from within Lisp as well as an easy specification for non-Lisp collaboration partners when creating ontology components for inclusion into Omega.

5.3.1 Indexing Ontology Vocabularies

Since Omega concepts are Lisp symbols segregated into role-specific packages, we can simply use APROPOS (or APROPOS-LIST) to do substring search. For example, in the DINO browser, a user can request all words or concepts matching the string "popcorn", yielding the concepts |popcorn|, |popcorn<grain| and |popcorn ball|. As ontology size grows, the cost of APROPOS will grow linearly in the symbol base. To address this, we recently implemented a trigram-based indexing scheme to improve interactive performance. The word and concept spaces of a given ovocab are preprocessed to generate all containing trigrams (after first compressing the character set, by folding case, replacing accented letters with a "close" corresponding unaccented letter, and replacing all non-letters with a special 'other' symbol: the result is 27^3 or roughly 20,000 trigrams, of which in practice around 55% are non-empty). Each word and concept is indexed under each of its contained trigrams; the size of each trigram index is recorded as well.

When a substring request is presented to an ovocab indexed as above, the request is first preprocessed to determine the least common trigram it contains. (Searches where the substring is two or fewer characters are not common because of the large number of return values; they are implemented as a special case). Only the concepts or words indexed under that least common trigram need be searched (however, because of the character compression, each item must be examined, unlike the analogous case with APROPOS-LIST and with case-sensitive search requested). The result for is that these sorts of matches are sped up by a factor of up to 5 for sufficiently large ontologies. The chief disadvantage of this scheme is that

trigram references are indexed, not strings themselves; a word or concept name containing the same trigram twice (e.g., |martial art|) will be indexed twice.

The generation time and size of the index are moderate and seem well worth the performance speedup obtained. However, we are considering two other approaches to address resources needs for indexing.

- The character mapping scheme described above was chosen to minimize the number of index buckets (especially empty index buckets), but using only intuition to derive the particular mappings. A Huffman-style code might provide better cues as to more felicitous mappings, possibly improving the tradeoff between index size and discrimination. The cost of index generation of substring query preprocessing might increase slightly, however.
- The index itself, essentially a vector of size 27^3 , is not particularly useful *per se* in the interactive Lisp environment, and yet it does require non-trivial run-time resources. Because trigrams can be represented as base 27 integers, an alternative implementation would be to move the index only into a random-access file or database table; the returned set of candidate matches would be individually examined as before.

A final characteristic of the system which we attribute to it embedding in the Lisp environment is the substantial dependence on shared code from others in the Lisp community. Most notably, we have used for various purposes in our Omega-related research all of the following community and vendor-supplied code resources:

- AODBC and MaiSQL
- Allegroserve and Portable Allegroserve
- Mark Nahabedian's CL-WordNet interface package
- CLOCC, especially multiprocessing and socket support
- CMU Lisp Utilities, including time/date parser, MK-DEFSYSTEM
- Paul Graham's ATN parser

6. CURRENT AND FUTURE WORK

Omega is largely a mature artifact, although some work is ongoing, and a few new areas are under consideration. One major current area of interest is in adding large quantities of relatively highly structured information, such as flat hierarchies descending from a known root and small domain-specific encoded hierarchies, as well as named entities where the parent concept may be less clear. These include:

- 55,000 natural and political geographic locations, such as cities, ports, mountains, airfields, etc.
- 8000 institutions of higher education
- 90000 medium-to-lower named entities from the TAP KB [11]
- 2500 areas of commercial activity, from the US Census NAICS system [12]
- 440,000 named instances obtained by data mining from the WWW

In this vein, we are also considering incorporating all or part of the DMOZ classification structure, world airport databases,

publicly traded companies, etc. We anticipate that scalability and cross-ontology merging, and disambiguation of named entity references will be significant issues.

Another area of current investigation regards the multi-perspective "shishkebab" conceptualization discussed above. We hope to articulate or even learn common patterns of shishkebabs from identified examples, and use them to anticipate and even suggest new shishkebab senses; for example, a newly acquired ontology may use a term in a way which is unanticipated but is similar to an shishkebab schema attached to an closely related concept. To date, however, results have been mixed.

Finally, we anticipate that in the coming months we will reformulate Omega using the newly released WordNet-2.0 and investigate reformulating Omega for the Semantic Web.

7. ACKNOWLEDGMENTS

We would especially like to thank Kevin Knight, Richard Whitney, Ramesh Patil, Bill Swartout, Jonathan Graehl for the original Sensus, Sensus Ontosaurus, and associated tools they created. Our thanks go also to Kavi Mahesh and Sergei Nirenburg for the providing and assisting with Mikrokosmos. We thank Scott Smith, Sid Berkowitz, and Boyan Onyshkevych for their support, encouragement, testing and suggestions regarding various releases of Omega.

8. REFERENCES

- [1] Bateman, J.A., Kasper, R.T., Moore, J.D., and Whitney, R.A. (1989). A General Organization of Knowledge for Natural Language Processing: The Penman Upper Model. Unpublished research report, USC/Information Sciences Institute, Marina del Rey, CA.
- [2] Fellbaum, C. (ed.) WordNet: An On-line Lexical Database and Some of its Applications. MIT Press, Cambridge, MA, 1998.
- [3] Hovy, E. H. Combining and Standardizing Large-Scale, Practical Ontologies for Machine Translation and Other Uses, in Proceedings of LREC (Granada, Spain, 1998).
- [4] Hovy, E. H., Philpot, A., Ambite, J. L., Arens, Y., Klavans, J., Bourne, W., and Saroz, D. Data Acquisition and Integration in the DGRC's Energy Data Collection Project, in Proceedings of the NSF's dg.o 2001 (Los Angeles, CA, May 2001).
- [5] Hovy, E., Philpot, A., Klavans, J., Germann, U., Davis, P., and Popper S. Extending Metadata Definitions by Automatically Extracting and Organizing Glossary Definitions, in Proceedings of the NSF's dg.o 2003 (Boston, MA, May 2003).
- [6] Klavans, J. L., Davis, P. T., and Popper, S. Building Large Ontologies Using Web-Crawling and Glossary Analysis Techniques, in Proceedings of the NSF's dg.o 2002 (Los Angeles, CA, May 2002).
- [7] Knight, K, and Luk, S. K. Building a Large-Scale Knowledge Base for Machine Translation. Proceedings of AAAI (Seattle, WA, 1994)
- [8] Lenat, D. B. CYC: A Large-Scale Investment in Knowledge Infrastructure". Communications of the ACM, 38, 11, 33-38.
- [9] Mahesh, K., and Nirenberg, S. A Situated Ontology for Practical NLP, in Proceedings on the Workshop on Basic Ontological Issues in Knowledge Sharing at IJCAI-95. (Montreal, Canada, 1995).
- [10] Philpot, A., Ambite, J. L., and Hovy, E. H. DGRC AskCal: Natural Language Question Answering for Energy Time Series, in Proceedings of the NSF's dg.o 2002 (Los Angeles, CA, May 2002).
- [11] The Syntax of CycL. <http://www.cyc.com/cycl.html>
<http://www.acm.org/sigs/pubs/proceed/template.html>.
- [12] The TAP KB. <http://tap.stanford.edu>
- [13] North American Industry Classification System. <http://www.census.gov/epcd/www/naics.html>