

# Overview of Approach, Methodologies, Standards, and Tools for Ontologies

\* \* \* DRAFT \* \* \*

Howard Beck  
Agricultural and Biological Engineering  
University of Florida  
hwb@agen.ufl.edu

Helena Sofia Pinto  
Department of Information Systems  
and Computer Science  
Universidade Técnica de Lisboa  
sofia@gia.ist.utl.pt

The Agricultural Ontology Service  
UN FAO

## 1. The Problem to be Solved

*In this section we give a justification and suggest ways that ontologies can be used to better organize information resources and assist users in retrieving relevant information. Ontologies are contrasted with conventional search methods based on fulltext search, and the relationship between ontologies and thesauri is introduced. Applications of ontology in databases and natural language processing are also introduced.*

### The Searching Problem

Everyone knows the Internet has exploded with useful information, but nobody can find what they want, or so goes the claim that has led to the current interest in the Semantic Web [W3C 2001] and, in particular, the use of ontologies for organizing large collections of knowledge. The agricultural domain is no different from any other in that large repositories of knowledge are being created by thousands of individuals building Web sites around the world. Finding information within a single site can be difficult, whereas searching for information across multiple sites at different institutions, which are probably written in different languages, can be an overwhelming task. Of course, the agricultural domain contains concepts which are unique to agriculture. Ontologies attempt to exploit domain-specific information by representing the meaning of terms within a domain, and using these meaning representations to organize the collection and make search more accurate. Exactly how meaning is represented, how to organize collections around this representational framework, and how search and other inference operations are implemented are all technical issues related to ontology construction. This paper illustrates how problems of information organization and search in the agricultural domain can be addressed by using ontologies, and presents a brief overview of the methodologies, standards, and tools available for this task.

To be sure, conventional information retrieval technologies, consisting mainly of variations on fulltext search which is the basis of well known search engines such as Google [2002], Lycos [2002], and AltaVista [2002] have performed remarkably well, and are currently the method of choice for searching the Web. In classic information retrieval

[Salton and McGill 1983], statistical techniques are used based on the frequency of key words appearing in a document. In its simplest form, a fulltext search engine contains an index of every word occurring in every document within the collection to be searched. For example, the word “crop” would appear in this index, and would point to every document containing the word “crop”. A user searching for “crop” is shown a list of all these documents, along with a ranking based on the number of times the word “crop” appeared in each document (presumably the more times the word appears in a document, the more relevant that document is to the user doing the search).

Because there are so many documents on the Web, the chances of finding something containing a particular word or phrase is quite good. It is so good in fact that conventional search engines typically identify hundreds of thousands of documents matching a user’s search terms. And that is the main problem with conventional search. The overwhelming number of these documents are not relevant to the user’s interest. Perhaps more dangerous is the possibility that documents do exist that are very important to the user, but they were not identified, usually because different words were used in the document that did not match the search terms directly. These two types of errors are formally measured as precision and recall:

$$\text{Precision} = \frac{\text{Number of Relevant Documents Identified}}{\text{Total Number of Documents Identified}}$$

$$\text{Recall} = \frac{\text{Number of Relevant Documents Identified}}{\text{Number of Relevant Documents in the Collection}}$$

A search engine with perfect precision and recall would find all and only the documents relevant to the user’s interest. Precision is a measure of how many of the documents identified as a result of a search are relevant to the user. Typically with fulltext search most are not, and precision can be as low as 5 percent. Recall is a measure of how well the search engine did in locating relevant documents (did it find all the relevant documents in the collection). Again, recall statistics can be quite low for even the best search engines.

Furthermore, fulltext search is not capable of processing structured queries, such as “list the countries that produce cassava”, since they would simply produce a list of documents containing these terms. Preexisting text publications would not be structured in a way that can answer this question. Processing this query correctly would require a database that explicitly represents facts about agricultural production practices by country.

Fulltext search based only on statistical methods makes no attempt to understand the meaning of the terms being used. This is the main cause of poor performance. Words can have many senses (food bank verses river bank), synonymous terms are used in different situations (farmer verses grower), words have a wide variety of different associations and interrelationships (peanut is a kind of crop, leaf is part of a plant) and terms appear in different languages (peanut in English, cacahuete in Spanish). A big

improvement in search precision and recall could be obtained if these relationships could be taken into consideration, and that is what ontologies attempt to do.

## **Thesaurus**

For well over a century, librarians have made use of thesauri for building subject classifications and cataloging documents within subject headings. The relevance of the thesaurus to the electronic information age is now being realized. The thesaurus can be considered an early, although simple, kind of ontology, and. A thesaurus attempts to categorize subject terms using a variety of simple abstractions, the main ones being:

Broader Term (BT) - A particular term is more general than another term (“crop” is broader than “soybeans”)

Narrower Term (NT) – A particular term is more specific than another (“soybeans” is narrower than “crop”)

Related Term (RT) – Two terms are associated (“leaf” is related to “plant”)

Use For (UF) – A particular term is the preferred term among a set of synonymous terms (use “grower” for “farmer”)

Thus, a thesaurus entry for the term “soybean” might look like:

Soybean

BT: legume

NT: Bragg, Cobb, ...

RT: pod, leaf

UF: soy

The BT/NT relationships give rise to taxonomies which organizing terms in hierarchical categories. Taxonomies are an important component of both thesauri and ontologies.

The thesaurus provides a structured representation among terms in a domain, hence it is a kind of meaning representation. The advantages gained by this approach are that users can search directly for information that has been manually cataloged within these subject headings, and that associations to related but different terms can be used to navigate within a neighborhood of relevant topics. Thus thesaurus-based search can have a high precision rate, and recall is generally improved over fulltext search. Several well-know agricultural thesauri [AGROVOC 2002, CABI 2002, NAL 2002] have been constructed, some of them have been in use for many decades.

While similar in general structure, ontologies attempt to do an even better job than thesauri. The main way they do this is by providing a more detailed, formal knowledge representation language that does a more thorough of representing word meaning. While BT/NT/RT/UF relationships are simple and useful, they can be vague, and certainly don't cover all the rich ways in which words can be interrelated.

## **Ontologies**

Here is a simple language for building an ontology that improves slightly on the basic thesaurus abstractions<sup>1</sup>:

- Class: A generic concept
- Object: A particular occurrence of a generic concept
- Subclass: A class that is more specific than a particular class
- Superclass: A class that is more general than a particular class
- PartOf: An object that is part of a particular object
- Association: Two objects are related in general (other than one of the above relationships)

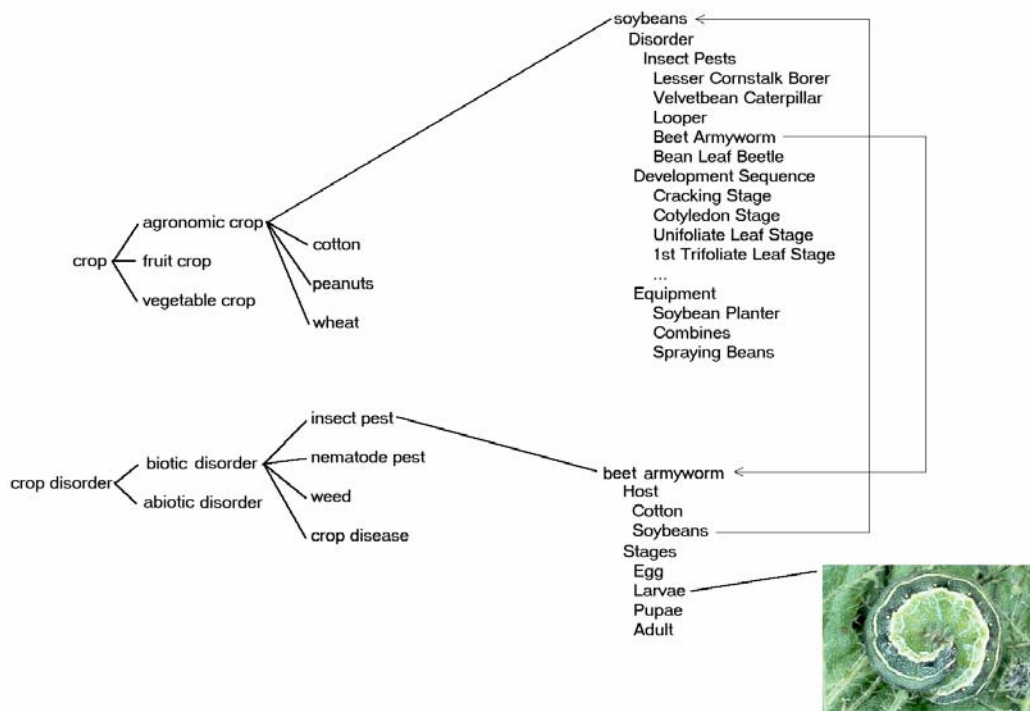


Figure 1. Sample ontology for crop-pest management.

A sample ontology of the crop-pest domain is shown in Figure 1, and illustrates some of these simple relationships. “Beet Armyworm” (Note, this would be an actual occurrence of beet armyworm) is an object within the “insect pest” class. There are several taxonomic superclass-class-subclass relationships, for example “crop” – “agronomic crop” – “soybeans”. “Beet Armyworm” is associated with “Soybeans”, specifically it is an “Insect Pest” of soybeans.

<sup>1</sup> This is intended only as an introductory example. Formal languages for building ontologies are presented in section 4.

## **How does this improve search?**

Equipped with such a representation containing interconnections between related terms, the search engine now has information about the meaning of terms that can be useful. Instead of the term “Soybean” being treated as a string, and counting the frequency with which this string occurs within a document, the search engine can exploit the term relationships to get directly to information about soybeans. The user enters the term “Soybeans” and jumps directly to the node for this concept shown in Figure 1. All information directly related to this concept is directly accessible via the term relationships. Moreover, it can use these relations to examine similar or related concepts. In this way, the user gets access to all and only the information available on a particular concept.

## **Issues in Database Management and Natural Language Processing**

The ontology acts as a framework for organizing the concepts within a domain. In addition, information resources such as documents can be attached to concepts, a process known as cataloging. Traditionally cataloging is a labor-intensive manual process, requiring special training. Tools for automating or semi-automating this process are much in demand, but existing tools do not perform as well as human catalogers.

Attaching information resources to ontologies creates a complete database, the result being that users can perform queries to retrieve specific information. Ontologies can attach to existing database management systems, or even ad hoc files containing documents, photographs, video, or other media. Typically the attached information resources are considered to be outside the ontology. Alternatively, some knowledge representation languages used to build ontologies have advanced to the point where they can act as complete data modeling languages, and databases can be constructed directly within the ontology, such as CLASSIC [Borgida et al. 1998] and ConceptBase [Jeusfeld et al. 1998]. In such systems the distinction between ontology and database is blurred.

Ideally the user could express queries in natural language, such as “List all insects that damage soybean leaves”, or “What are the vegetative stages of soybean development”, and instead of getting a list of documents which the user must read to find the answer, a direct reply would be generated by the system. The steps involved in natural language query processing include finding each word in the query in a dictionary or lexicon, analyzing the grammatical patterns within the stated phrase or sentence (syntactic analysis), mapping the grammatical structure onto objects in the ontology (semantic analysis), and then drawing inferences on the relationships between the query and other objects in the database (query processing), the result being a precise retrieval of objects matching the user’s interest.

Other uses for natural language processing (NLP) include information extract, and machine translation. In information extraction, NLP techniques are used to automatically extract facts from plain text. Machine translation has obvious uses in converting document written in one language, such as Spanish, into another language, such as

English. Natural language processing is an extremely difficult area, yet the ontology promises to provide an important facility necessary for the construction of natural language systems by providing a representation for the meaning of concepts in a domain.

Section 2 gives a more formal definition of ontologies, what they look like, and levels of representation running from abstract to specific within the same ontology. Methodologies for construction of ontologies, which is a labor-intensive task, are addressed in Section 3, including issues such as merging existing ontologies and version control. Section 4 gives an overview of formal languages used for representing ontologies. This includes an historical view of semantic networks, and recent developments to define languages for constructing ontologies within the new XML (Extensible Markup Language) standards. Section 5 gives a brief overview of available commercial and public domain tools that assist in ontology construction and use. Section 6 discusses advanced topics, including database management and natural language processing, in greater detail. .

## 2. What are Ontologies?

*In this section we discuss what ontologies are. First, we give a general idea of what ontologies are. Then, we discuss the differences between ontologies and knowledge bases and between ontologies and thesauri. Then, we present and discuss the different definitions of “ontology” and the different forms that they may take. Finally, we analyze the different types of ontologies, and indicate ways to address multilingual issues.*

### General Idea

Ontologies have been proposed to solve the problems that arise from using different terminology to refer to the same concept or using the same term to refer to different concepts. The term “ontology” has been borrowed from Philosophy. In Knowledge Sharing [Neches et al. 1991, Patileta 1992, Swartout 1994] the meaning of this word is different from its meaning in Philosophy. Gruber [1993] introduced the term *ontology* to mean an “explicit specification of a conceptualization” while in Philosophy *Ontology* means “a systematic account of Existence”.<sup>2</sup> To distinguish between both meanings [Guarino and Giarretta 1995] proposed that *Ontology* (upper-case “o”) should refer to the Philosophy meaning and *ontology* (lower-case “o”) to the AI meaning.

---

<sup>2</sup> “Ontology is the branch of Philosophy which deals with the nature and organization of reality. Aristotle defined it as the science of being as such.” [Guarino and Giarretta 1995]. The word *Ontology* was initially a synonym for *Metaphysics*. However, it has been divided into on the one hand the study of the essence of beings and on the other hand the study and definition of a formal theory of the objects, that is, the study of the basic characteristics of the whole reality.

An ontology specifies common vocabulary between different systems. It tries to identify and overcome the barriers to sharing and reuse of knowledge represented by AI programs that are due to a lack of consensus in what regards the vocabulary used and the different semantic interpretations in domain models. Informally, an ontology consists of a set of terms and a set of constraints imposed on the way those terms can be combined. The latter set constrains the semantics of a term since it restricts the number of possible interpretations of the term. Terms in an ontology are a representation of concepts. We should stress that in an ontology concepts are represented, not words. Concepts, in general, are not specific of a given natural language [Mars 1995].

Ontologies are closely related to knowledge bases. The distinction between ontologies and knowledge bases lies on the different role played by represented knowledge. Ontologies tend to represent knowledge that is more or less consensual of a community of people, whereas knowledge bases represent knowledge that is specific of the particular problem that the knowledge based system solves. Ontologies are concerned with static domain knowledge. A knowledge base usually includes knowledge that changes with inferences. Knowledge represented in ontologies does not change with inference. For instance, while an ontology on enterprise modeling contains concepts, such as activity, process, resource, in a knowledge base one would have represented the particular activities that are performed by a particular enterprise, the particular processes that take place in that enterprise, the actual process, activities, costs, resources that were used to build or produce a particular product, an estimate of the resources that were inferred to be needed to satisfy a new order that has just arrived. Therefore, knowledge in ontologies is more appropriate to be reused and shared across applications.

Although ontologies aim at capturing static domain knowledge, it is generally acknowledged that an ontology depends on the application that powered its construction. If two applications are dealing with the same domain but the tasks they have to perform are different, then it is natural that the ontologies they need about that domain are slightly different. Although most of the concepts are usually common they may be defined in different ways, such as with different levels of detail (as a class, a relation, etc.), capturing different points of view or features about the same concept (from a structural point of view, a functional point of view, etc.), with different levels of granularity. Different points of view may also imply that the same concepts are represented using different terminology. It should be stressed that there is no single way of organizing concepts. There are different genuine alternatives. Therefore, one commits itself when an alternative is chosen.

Ontologies play an important part in communication between intelligent systems. Suppose that one application asks the other to perform a task. While transmitting information about the particular problem that it wants to see solved it must transmit that information in such a way that the other application can understand. Therefore, it may be important to translate information between different ontologies about the same domain.

Not only is compatibility among ontologies important in communication between different applications, but it is also important in building large systems from smaller

ones. Any knowledge based system has an ontology as one of its components, even if only implicitly. A knowledge base can only be assembled from smaller ones if their underlying ontologies are compatible and consistent one with the others. If the ontologies underlying the different knowledge bases are not the same it means that either the domain is represented using different terminology or the same terminology is used with different meaning, that is, either the terms in each ontology are different or the axioms representing the constraints imposed on those terms are not equivalent. Only if the underlying ontologies of the knowledge bases are the same can they be assembled together. Therefore, one cannot build a large system by means of reuse if there is no understanding in what concerns the vocabulary that is used or if there is understanding about the vocabulary that is used but the axioms don't represent the same statements or are in contradiction with one another.

### **What Is the Difference Between an Ontology and a Thesaurus?**

Given the general definition of ontology stated above, it is important to ask, what is the difference between an ontology and a thesaurus? Many potential ontology users and partners in the library sciences are very familiar with the thesaurus, and make uses of a thesaurus in cataloging information. They will want to know, what is the advantage in moving from a thesaurus to an ontology?

Because the above definition of ontology is very general, it might be argued that a thesaurus *is* an ontology. There are features in a thesaurus that are common to ontological theories, but others that aren't. The common features include organization of terminology and hierarchical structure. Both an ontology and thesaurus are concerned about covering a broad range of terminology used in a particular domain, and in understanding the relationships among these terms. Both utilize a hierarchical organization to group terms into categories and subcategories. Both can be applied to cataloging and organizing information. Important differences include informality and ambiguity of relations in a thesaurus. The relationships (BT/NT/RT/UF, etc) available for organizing the terms in a thesaurus are not only relatively few in number, but are not formally defined and thus subject to ambiguous use. For example, the BT/NT (broader than/ narrower than) relationship can be used ambiguously to both indicate that a particular concept is a special case of another, or that a concept is part of another. The RT (related to) relationship covers all other relationships, lumping together associations, arbitrary properties, and other vague relationships. A good ontology can introduce a host of structural and conceptual relationships including superclass/subclass/instance relationships, property values, time relationships, and others depending on the representation language used. In general an ontology contains far more relationships, which are formally defined and unambiguous, compared to a thesaurus.

Another way to look at this is to compare the goals of a thesaurus with the goals of an ontology. A thesaurus attempts to show the relationships between *terms*, whereas an ontology attempts to define *concepts* and show the relationships between concepts. In pure form, an ontology is not about terms at all, only about concepts which ideally are represented in a form independent of terms in any natural language. A thesaurus makes

no attempt to define, let alone formally represent, the meaning of concepts, it is concerned only with relationships among terms in a particular natural language (or multiple natural languages if the thesaurus is multilingual). Thus the machinery for representing concepts in an ontology must be much stronger. Nevertheless to talk about an ontology, we must use terms in some natural language, so the ontology must include a mapping from terms to concepts, no such mapping is formally recognized in a thesaurus.

In practical applications, this distinction implies that an ontology will better than a thesaurus when it comes to searching. Because the ontology contains machine interpretable definitions of concepts, it is able to support terminological reasoning. This means that a user's question can be understood through analysis of the meaning of the user's terms appearing in the question, and mapped more precisely to information resources. The ontology can reason about the meaning of concepts by comparing logical concept structures. A simple example (see section) is subsumption. An ontology can reason that one concept is a special case of another because the logical definitions of each concept can be compared. If concept B satisfies the requirements for being a case of concept A, then B can automatically be classified below A. This gives rise to query processing and searching which is not possible with a thesaurus.

### **Ontology definitions**

The term "ontology" has been used in AI with several meanings. A discussion of some of the meanings in Philosophy, in AI, and in the Knowledge Sharing area can be found in [Guarino and Giaretta 1995]. The initial definition proposed by Gruber [1993] was slightly modified in [Borst 1997]. A merge of both definitions can be phrased as:<sup>3</sup>

“an explicit formal specification of a shared conceptualization”

As discussed in [Studer et al.1998]:

- “explicit” means that “the type of concepts used and the constraints on their use are explicitly defined”;
- “formal” refers to the fact that “it should be machine readable”;
- “shared” reflects the notion that the knowledge represented in an ontology “captures consensual knowledge, that is, it is not private to some individual, but accepted by a group”;
- “conceptualization” refers to “an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon”.<sup>4</sup>

---

<sup>3</sup> Ontologies are defined as “a formal specification of a shared conceptualization” in [Borst 1997].

<sup>4</sup> As pointed out in Guarino and Giaretta 1995,], conceptualization in this definition should not be understood as introduced in [Genesereth et al. 1987]. Genesereth &

On the other end of possible ontology definitions, the broadest, more informal, definition of ontology is [Uschold et al. 1996]:

“a vocabulary of terms and some specification of their meaning”

The main differences between both definitions are the formality requirement and the consensual nature of the knowledge represented in an ontology. It is important that the knowledge represented in the ontology has a consensual nature, at least among a given group, so that it can be reused in several knowledge based systems. After all, this is the main reason why ontologies are built. The formality requirement is not consensual. There are ontologies which are expressed in a restricted and structured form of natural language that are nonetheless considered ontologies, for instance, the text version of an ontology about activities, processes, organization, strategy [Uschold et al. 1998]. There are even ontologies which are loosely expressed in natural language [Uschold et al. 1996].

### **What do they look like?**

An ontology usually takes the form of an hierarchy of symbols. The symbols represent the concepts of a particular domain. Sometimes the hierarchy is referred to as a taxonomy and symbols are referred to as concepts, vocabulary or terms. However, this is not enough since these constituents could be interpreted differently by different systems. To restrict the possible interpretations of its symbols, an ontology includes a set of axioms. These axioms express the constraints that the symbols involved in those axioms must comply to. These axioms relate one symbol<sup>5</sup> with the other symbols of the ontology. They restrict the possible interpretations for that symbol. Therefore, the most important part of an ontology is the semantics associated with its symbols, usually referred to as the content of the ontology.<sup>6</sup> The content of an ontology is constrained through its set of axioms. Therefore, the basic unit of meaning is not a symbol but the theory, that is, the set of axioms that is associated with the several symbols in the hierarchy.

To give an idea of what an ontology looks like, we present the definitions of the same few concepts from an existing ontology in both an informal and a formal way. In Figure 2 we show the text definition of an *activity* and a *doer* in the ENTERPRISE ontology [Uschold et al. 1998]. An activity is characterized by the interval during which the activity takes place, its pre-conditions (what must be true for the activity to be performed), its effects (what is true once the activity is completed). There are also other

---

Nilsson's notion of a conceptualization, as a structure containing the set of all objects of the universe of discourse and the set of relevant relations among those objects, is not appropriate due to the fact that both the objects and the relations are considered extensional entities. Therefore, their definition of a conceptualization is more appropriate to represent a state of affairs.

<sup>5</sup> That is being constrained.

<sup>6</sup> In opposition to form that usually is associated with its syntax.

attributes that characterize it, such as its doer, the sub-activities into which it can be decomposed. A doer is an actor that performs an activity. All concepts in upper case are also defined in the ontology. The definitions are expressed in a restricted and structured form of natural language.

ACTIVITY: something done over a particular TIME INTERVAL. The following may pertain to an ACTIVITY:

- has PRE-CONDITION(S);
- has EFFECT(S);
- is performed by one or more DOERS;
- is decomposed into more detailed SUB-ACTIVITIES
- entails use and/or consumption of RESOURCES
- has AUTHORITY requirements
- is associated with an [ACTIVITY] OWNER
- has a measured efficiency

DOER: The Role of an Actor in a Relationship with an ACTIVITY whereby the Actor performs (all or part of) the ACTIVITY.

Figure 2: Informal definitions in the ENTERPRISE ontology

In Figure 3 we present the definitions of those concepts implemented in Ontolingua and kept in the Ontolingua Server<sup>7</sup> library as Enterprise-Ontology. `Activity` and `actual-doer` (which corresponds to the concept of doer) were represented as classes.<sup>8</sup> As it can be seen, there may be some differences between the implemented and the textual versions of an ontology. Activity is characterized by the interval of time during which it takes place, its pre-conditions, its effects and its status (done, to be done, etc.). Activity and actual-doer are related by the `actually-execute` relation. The actual-doer is an actor for which there is an activity to which that actor is related by the `actually-execute` relation. In this case the relationships used to establish the hierarchy of concepts are class-superclass and instance-class, for instance, actual-doer is a subclass of actor.

---

<sup>7</sup> <http://WWW-KSL-SVC.stanford.edu:5915>

<sup>8</sup> Frames that are instances of `Class`.

```

(Define-Frame Activity
:Own-Slots
((Documentation "Something done over a particular Time-Range.
The following may pertain to an Activity:
* is performed by one or more Actual-Doer(s);
* is decomposed into more detailed Sub-Activity(s);
* Can-Use-Resource(s);
* An Actor may Hold-Authority to perform it;
* there may be an Activity-Owner;
* has a measured efficiency.")

(Instance-Of Class)
(Subclass-Of Activity-Or-Spec))
:Template-Slots ((Actual-Activity-Interval (Minimum-Cardinality 0)
                                           (Cardinality 1)
                                           (Value-Type Time-Range))
                 (Actual-Pre-Condition (Minimum-Cardinality 1)
                                       (Value-Type Pre-Condition))
                 (Actual-Effect (Minimum-Cardinality 1)
                                (Value-Type Effect))
                 (Activity-Status (Minimum-Cardinality 1)
                                  (Value-Type Activity-State))))

(Define-Frame Actual-Doer
:Own-Slots
((Documentation "The Actor in the Actually-Execute relationship.")
(Instance-Of Class)
(Subclass-Of Actor))
:Axioms (<=> (Actual-Doer ?Actor)
            (And (Actor ?Actor)
                 (Exists (?Activity)
                        (Actually-Execute ?Actor ?Activity))))

(Define-Relation Actually-Execute (?Actor ?Activity)
"A relationship between an Actor and an Activity
whereby the Actor has performed the Activity."
:Def (And (Potential-Actor ?Actor) (Activity ?Activity)))

```

Figure 3 Formal definitions in the ENTERPRISE ontology

## Types of Ontologies

Ontologies can be classified, according to the issue of the conceptualization, usually referred as type, into [van Heist et al. 1997, Guarino 1998]:

- *representation ontologies or meta-ontologies*, capture the representation primitives used to formalize knowledge in a given knowledge representation family or system. For instance, the Frame ontology [Gruber 1993] which defines the terms that capture conventions used in object-centered knowledge representation systems (frames, description logics, etc.). This ontology defines concepts, such as class, relation, function, named-axiom, arity, exact-domain, exact-range, unary-relation, binary-relation. In this ontology, relations are sets of tuples (named by predicates), functions are a special case of relations, classes are

- unary relations (there is no special syntax for types), there is no special treatment of slots (since they can be represented as unary functions or binary relations) and classes are defined extensionally as sets (not descriptions);
- *general or upper-level ontologies*,<sup>9</sup> classifies the different categories of entities existing in the world. Very general notions which are independent of a particular problem or domain are represented in these ontologies. Knowledge defined in this kind of ontologies is applicable across domains and includes vocabulary related to things, events, time, space, mereology.<sup>10</sup>

An ontology of time, Simple-Time, can be found in the Ontolingua Server library. In Figure 4 we present part of its structure and in Figure 5 we present some of the definitions of the concepts represented in it. There are three main concepts: time point, time range and duration. A `time point` defines a single point in time that cannot be further decomposed. A `time range` is characterized by two time points, its starting and ending time points, and a duration. The ending time of a time range is equal to the sum of its starting time and its duration. A `duration` denotes a period of time and is characterized by a value and a measure. One can define an equality relation between two time points (if they represent the same time point) or two time ranges (if they have the same starting points and the same ending points). One can define several relations between time ranges, such as before, after, meets, overlaps.

Other examples of upper-level ontologies can be found in [Sowa 2000]. Some of the abstract upper-level ontologies presented in it were proposed by philosophers (for instance, Aristotle's ontology) and others by knowledge engineers (for instance, Cyc's [Lenat and Guha 1990] upper-level);

---

<sup>9</sup> They are also referred to as common/generic/top-level ontologies.

<sup>10</sup> This ontology deals with the spatio-material properties of the physical objects, such as the relation part-of, proper overlap, interior part [Borgo et al. 1996].

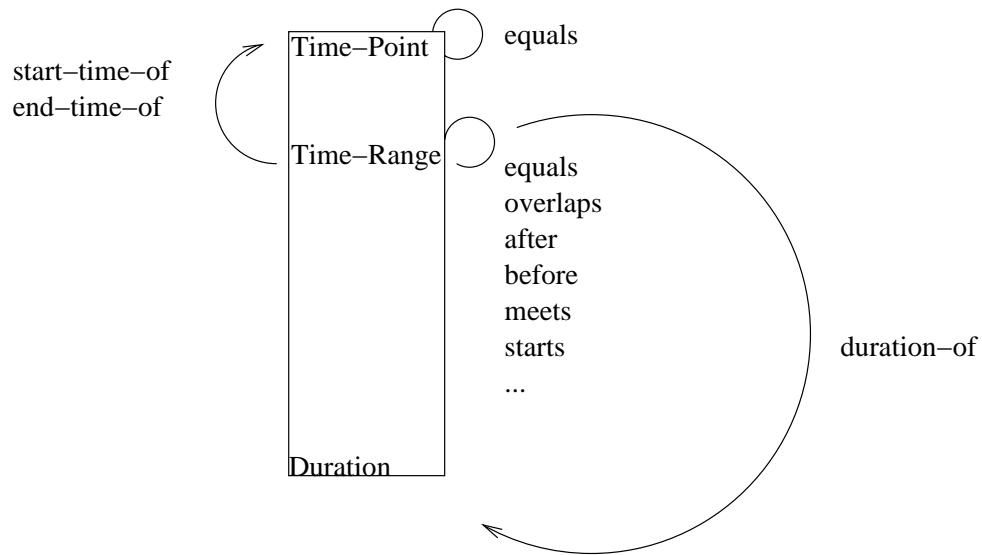


Figure 4: Part of the structure of Simple-Time ontology

```
(Define-Frame Time-Point
:Own-Slots
((Arity 1)
(Documentation "A time-point is a point in real, historical
time (on earth). It is independent of observer and context....
The time-points at which events occur can be known with various
degrees of precision and approximation, but conceptually time-
points
are point-like and not interval-like....")
(Domain-Of Day-of Minutes-of Hour-of Month-of Seconds-of Year-of)
(Instance-Of Class)
(Subclass-Of individual)))

(Define-Class Time-Range (?time-range)
"Time-Range denotes a certain period of time. It consists of a
start time, an end time. A start time must proceed an end time.
Relations between Time-Ranges are defined after James Allen's
interval
relations."
:def (individual ?time-range)
:constraints (Equals (+ (Start-Time-of ?time-range)
(Duration-of ?time-range))
(End-Time-of ?time-range)))

(Define-Function Start-Time-of (?time-range) :-> ?time-point
"(Start-Time-of 'tr) denotes a start time of a time range tr."
:def (and (Time-Range ?time-range)
(Time-Point ?time-point))

(Define-Class Duration (?duration)
"Duration denotes a period of time. It consists of a value and a
measure"
:def (individual ?duration))

(Define-Relation Equals (?t1 ?t2)
```

```

" a time point ?t1 is equal to a time point ?t2.
  a time range ?t1 is identical to a time range ?t2."
:axiom-def
((=> (and (Time-Point ?t1) (Time-Point ?t2))
      (<=> (Equals ?t1 ?t2)
            (and (= (Year-of ?t1) (Year-of ?t2))
                  (= (Month-of ?t1) (Month-of ?t2))
                  (= (Day-of ?t1) (Day-of ?t2))
                  (= (Hour-of ?t1) (Hour-of ?t2))
                  (= (Minute-of ?t1) (Minute-of ?t2))
                  (= (Second-of ?t1) (Second-of ?t2))))))
(=> (and (Time-Range ?t1) (Time-Range ?t2))
      (<=> (Equals ?t1 ?t2)
            (and (Equals (Start-Time-of ?t1)
                          (Start-Time-of ?t2))
                  (Equals (End-Time-of ?t1)
                          (End-Time-of ?t2))))))

(Define-Relation After (?time-range-1 ?time-range-2)
"a time range ?time-range-1 succeeds a time range ?time-range-2."
:iff-def (< (End-Time-of ?time-range-2)
            (Start-Time-of ?time-range-1))
:equivalent (Before ?time-range-2 ?time-range-1))

```

Figure 5: Some definitions from the Simple-Time ontology

- *domain ontologies*, are more specific ontologies. Knowledge represented in this kind of ontologies is specific to a particular domain. These ontologies describe vocabulary related to a generic domain, such as airplanes, chemical elements, etc. They provide vocabularies about concepts in a domain and their relationships or about the theories governing the domain. For instance, the Plinius ontology [van der Vet et al. 1995, Speel 1995] is about the chemical composition of ceramic materials and Chemical-Elements [Mariano 1996] is an ontology about the chemical elements.

In Chemical-Elements, the most general concept, `Elements`, is characterized by its symbol, atomic number, chemical group, chemical period, atomic weight, boiling point, melting point, crystal structure, density at 20 degrees centigrade, electronegativity, etc. Ontologies are usually hierarchically organized. In Figure 6 we present part of the hierarchy of Chemical-Elements. Dashed lines represent instance-class relations whereas solid lines represent class-superclass relations. Elements are divided into non-reactive and reactive. Helium, neon, argon, etc. are instances of non-reactive elements. They are all gases at normal pressure and temperature conditions. Reactive elements are further divided into metals, semi-metals and non-metals. There are several non-metal<sup>11</sup> elements, such as carbon and oxygen. A few of them are grouped in the `halogens` subclass.

<sup>11</sup> They are characterized by having high electronegativity.

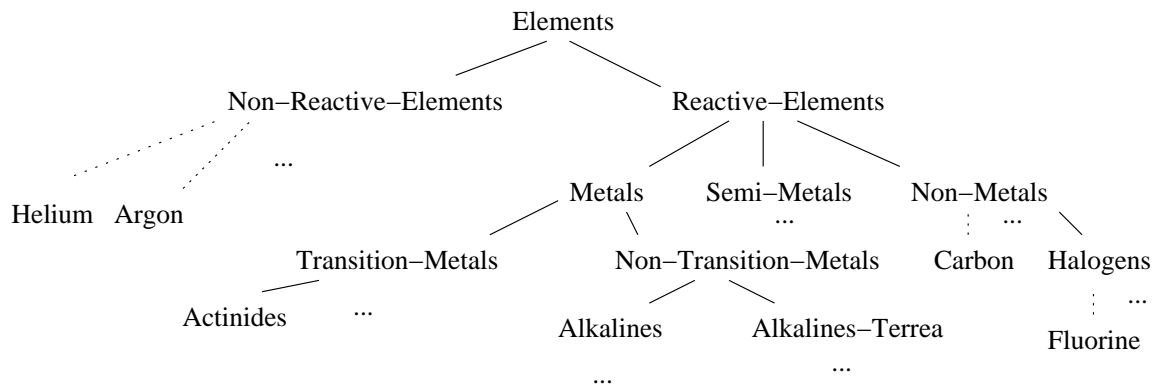


Figure 6: Part of Chemical-Elements hierarchy

- *application ontologies*, describe knowledge pieces depending both on a particular domain and task. Therefore, they are related to problem solving methods, which is outside the scope of this work. An application ontology relates concepts that describe the domain with concepts that are part of the description of problem solving methods. They explicit the role played by concepts of the domain in a given problem solving method.

In Figure 7 we present part of an example described in [Van Heist et al. 1997]. CASNET [Weiss et al. 1984] allows the representation of casual links that describe the processes associated to diseases and the development of diagnosis applications. Simple boxes represent domain knowledge and labeled boxes represent knowledge associated to problem solving methods. In this example, CASNET ontology defines pathophysiological states and observations. Dashed arrows represent class-superclass relations and labeled arrows represent the relation named by its label. In this example there are two inference methods: casnet abduction and casnet ranking. The application ontology relates concepts from the domain with concepts related to problem solving methods. Observations provide evidence for states which are associated with abduction confidence measures. One of the procedures involved in abduction uses the evidence links between observations and their associated confidence measures to compute the confidence measure of the state. One of the procedures involved in ranking uses the strengths of the casual relations between the states. The total weight of a state is the maximum of the forward and the inverse weights. The forward weight of a state summarizes the weight of the evidences coming from the causes of that state. The inverse weight summarizes the weight of the evidences coming from the effects of the state. The procedure that ranks states (hypothesis) uses the ratio of the weight of the hypothesis and the costs of testing that hypothesis.

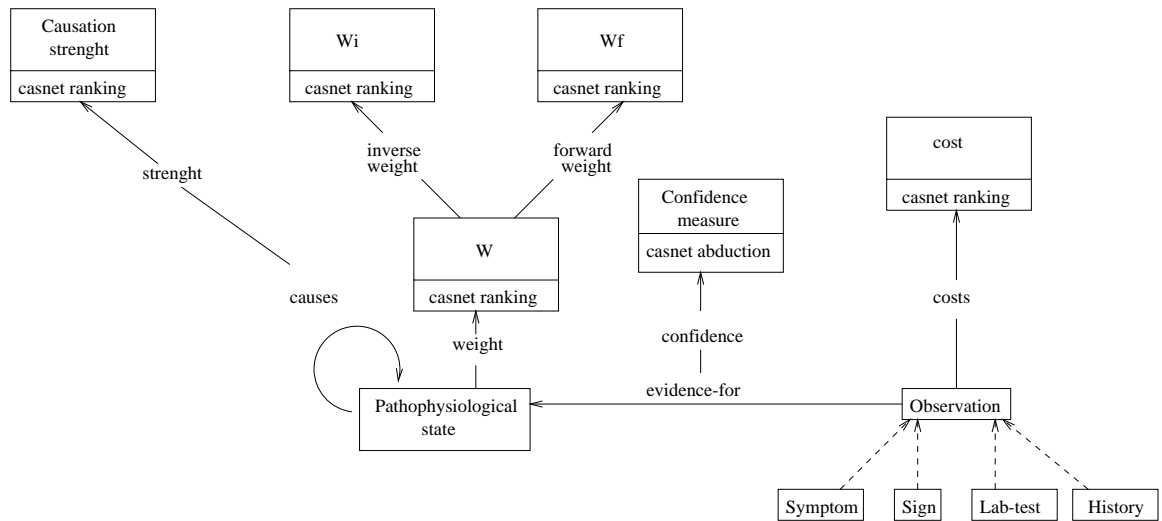


Figure 7: Part of an application ontology

Upper-level ontologies and domain ontologies are somewhat related since their aim is to represent a conceptualization of reality. General ontologies usually represent knowledge that is related or borrowed from Ontologies.<sup>12</sup> There is a large range of domain ontologies: general domain ontologies, such as about airplanes or chemical substances and more domain specific ontologies, such as about turbojet engines used in aircraft propulsion or about chemical elements. The distinction between domain ontologies and upper-level ontologies is not clear cut. In between the extremes one can see a middle ground. This middle ground specializes the upper-level and provides the upper-structure for domain ontologies. One can see this middle ground as representing general domain knowledge (general knowledge in one field, for instance medicine, physical engineering systems). Representation and application ontologies are orthogonal to domain and upper-level ontologies. Representation ontologies try to conceptualize the meta-level categories used to model the world (concept, property, etc.). Application ontologies connect ontologies with problem solving methods in a particular knowledge based system.

Although the previous classification is the most consensual in the area there are other proposals. For instance, [Mizoguchi et al. 1995] proposes a classification according to ontology content. In this case, only three kinds of ontologies are considered: *domain* ontologies, *general* ontologies and *task* ontologies. While the first two were already described, task ontologies provide the terms used to solve problems associated with a particular task. Therefore, they are related to problem solving methods, which is outside the scope of this work. For instance, in diagnosis the terms defined in the task ontology would include “observation”, “hypothesis” and “goal”.

<sup>12</sup> See [Sowa 2000] for some examples of useful Ontologies that provide knowledge that should be represented in upper-level ontologies.

## Strategies for Creating Multilingual Ontologies

A strategy is needed for developing an ontology in such a way that it can be used by people having different native (natural) languages, such as English, French, Japanese, and potentially many other languages. Furthermore, even within a general language there can be local dialects or even local communities of users who use terms in a way that differ from more general usage. We use the term “multilingual” to refer to the ability to support multiple, different natural languages (and to distinguish this from the separate issue of using different formal knowledge representation languages used for representing ontologies). Although an ontology represents concepts in a language-independent fashion, mappings from terms in natural language to concepts in the ontology enable people to access and interpret the concepts. The terms are expressed in the person’s natural language, and in a multilingual ontology, terms from many different natural languages are available. In addition, the multilingual aspect of an ontology can be beneficial in applications such as machine translation.

Two strategies for multilingual support in existing thesauri include the single concept approach and the interlingua approach. In the single concept approach, there is a single representation for a concept, and this representation includes a translation of the concept into terms in several different natural languages. In the interlingua approach, there is a multi-level representation involving a language-neutral (interlingua) concept representation, along with representations for each term in each supported natural language. Each natural language term is mapped to a single interlingua concept (in general each *sense* of the term is mapped to a single interlingua concept). Terms in different natural languages may map to the same interlingua concept, but it is not necessary that a given interlingua concept is mapped to by terms in all the languages. This is because a given concept can not necessarily be translated into a single term for all languages. Examples of these two approaches are presented below.

### 1. Single Concept Approach

#### AGROVOC

AGROVOC is a multilingual agricultural thesaurus developed by the United Nations Food and Agricultural Organization. A central AGROVOC server [AGROVOC, 2002] is available in English, French, Spanish and Portuguese, and versions in other languages are available or being prepared by national centers in other countries. The AGROVOC thesaurus contains the basic thesauri relationships (BT/NT/RT/UF). Each entry in the thesaurus contains terms for each language. Thus, the AGROVOC entry for “beef” contains:

BT meat  
  BT animal products  
RT veal

English: beef  
Spanish: carne de res  
French: viande bovine  
Portuguese: carne de bovino

A person making a new entry into AGROVOC is required to specify terms for all the supported languages. While this somewhat simplifies creation of multilingual terms in AGROVOC, the difficulty is that not all concepts have simple translations into a term in all the supported languages. This problem is solved, albeit at greater effort and complexity, in the interlingua approach.

## 2. Interlingua Approach

### EuroWordNet

EuroWordNet [Vossen 2001] is a multilingual ontology based on WordNet [Fellbaum, 1998] which is an English ontology developed at Princeton University. WordNet was developed first, and then EuroWordNet was developed later as a multilingual ontology. EuroWordNet is based on WordNet, and uses the same knowledge representation language, extended to provide multilingual support. The terms and concepts in WordNet are a subset of EuroWordNet (the English part).

WordNet provides a mechanism for dealing with synonyms and homonyms; different terms can have the same meaning, and the same term can have different meanings. A given term in WordNet is first broken into its different senses, which are also categorized by part-of-speech (noun, verb, adjective and adverb). Each sense of a term is mapped to a synset (synonym set) which is WordNet's representation for a concept. Different terms sharing a common sense are mapped to the same synset. The synset includes a gloss (short natural language definition of the concept, in English), and other conceptual information, such as pointers to antonym synsets.

In EuroWordNet, terms and synsets are created semi-independently for each natural language. It is not required that the concepts in one language are the same as that of another, though of course only concepts in the same domain can lead to a multilingual approach. Thus English can have its own terms and synsets and French can have its own terms and synsets, which allows for greater flexibility in describing language-specific concepts. Multilingual relationships are made at the level of synsets. An Inter-Lingual-Index (ILI) provides a mapping between synsets for the same concept in different language. Thus if synset in English represents the same concept as a synset in French, the English and French synsets would be linked by the ILI. The ILI is a way of indicating equivalence among synsets from different languages. Notice that the actual terms are not involved directly in this association. Also, an ILI does not necessarily link to a synset in all of the supported languages (some languages may not have terms for a particular concept).

The ILI is a kind of interlingua, however the ILI has no formal representational structure, it is merely a mechanism for linking equivalent synsets. Thus as an interlingua, the ILI provides comparatively weak ontological functions. Also, the ILI set in WordNet was initialized from the synsets from WordNet. Although the ILI set can be extended with non-English concepts, historically it is somewhat predisposed towards English.

EDR

To quote from the English introduction on EDR:

"The EDR Electronic Dictionary was developed for advanced processing of natural language by computers, and is composed of eleven sub-dictionaries. Sub-dictionaries include a concept dictionary, word dictionaries, bilingual dictionaries, etc. The EDR Electronic Dictionary is the result of a nine-year project (from fiscal 1986 to fiscal 1994) aimed at establishing an infrastructure for knowledge information processing. The project was funded by the Japan Key Technology Center and eight computer manufacturers (Fujitsu, Ltd., NEC Corporation, Hitachi, Ltd., Sharp Corporation, Toshiba Corporation, Oki Electric Industry Co., Ltd., Mitsubishi Electric Corporation, and Matsushita Electric Industrial Co., Ltd)" [<http://www.ijnet.or.jp/edr>]

The EDR Electronic Dictionary (Figure 8) is a machine-tractable dictionary that catalogues the lexical knowledge of Japanese and English and has unified thesaurus-like concept classifications with corpus databases. The Concept Classification Dictionary, a sub-dictionary of the Concept Dictionary, describes the similarity relation among concepts listed in the Word Dictionary. The EDR Corpus is the source for the information described in each of the sub-dictionaries. The basic approach taken during the development of the dictionary was to avoid a particular linguistic theory and to allow for adoptability to various applications." [Japan Electronic Dictionary Research Institute, Ltd, 1996]

EDR is bilingual (Japanese/English). Each concept has a hexadecimal identifier, and is primarily defined by its relationships to other concepts (these relationships are stored in the Concept Classification Dictionary), and by its links to one Japanese and/or one English word in the Word Dictionaries (see below). Each concept has a brief explanation in Japanese and/or English, intended to help those editing and maintaining the dictionary. The concept dictionary provides the interlingua, and concepts are modeled using a rich set of ontological relationships including agent, object, cause, material, occurrence, and time.

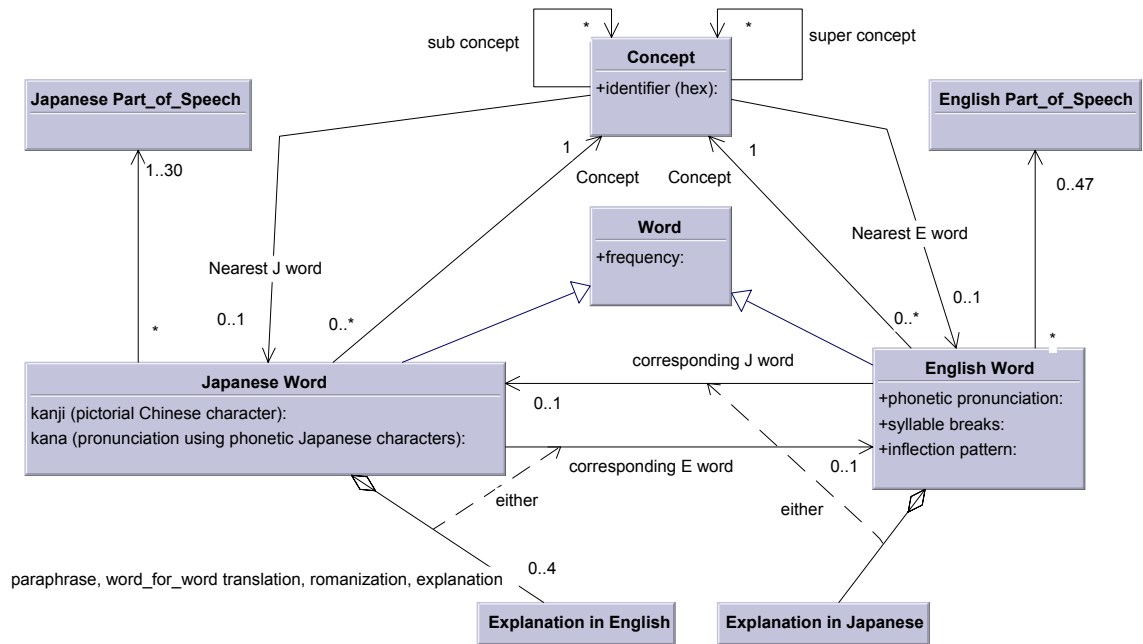


Figure 8. EDR structure

EDR is a fairly balanced bilingual dictionary, providing similar representation of both English and Japanese. The use of a language-neutral concept identifier is important in this regard. EDR attempts to handle the lack of corresponding words between languages using several kinds of translation. In the case of Japanese to English, the options are paraphrases (a phrase in English with equivalent meaning and implications), word-for-word translation, conversion of the Japanese word (in pictorial characters) into a Romanized form or explanation in English.

Note that the two languages have different requirements in terms of their “dictionary entries”. For example, inflection of Japanese verbs follows a few well-defined rules, so it is not necessary to record it, whereas English verb inflection is a nightmare (walk, walked; speak, spoke, think, thought; go, went...). Conversely, the pictorial characters used to represent Japanese usually have at least two pronunciations, and sometimes many, so guidance on the pronunciation is helpful. Note also that an EDR concept may have multiple “parents” – the structure is not a tree. There are also other relationships, other than BT/NT, which can be used to represent facts or occurrences.

### Character Sets

The Unicode standard [Unicode Consortium, 2000 #170], can represent most of the characters used in Asian languages. Although country-specific encodings are more commonly used on the Internet and in documents (eg SJIS and EUC in Japan), Unicode allows characters from many languages to be stored together. There isn't any obvious

alternative, and most countries will have mechanisms for converting from their own national encoding systems into Unicode. We would therefore propose that AOS use Unicode internally, with client applications converting to and from country-specific encodings as required.

### 3. How can ontologies be built?

*In this section we overview the way ontologies are built. First, we characterize the ontology building process and its life cycle. Then, we present an overview of the most representative methodologies to build ontologies from scratch. Then, we describe the processes of building ontologies by means of reuse. Finally, we describe the problems of ontology versioning.*

#### The ontology development life cycle

Ontology building is a process. A process is composed of a series of activities that are performed in order to achieve something. The usually accepted *stages* through which an ontology is built are:<sup>13</sup> specification, conceptualization, formalization, implementation, and maintenance. At each stage there are *activities* to be performed:

- *specification*, where one identifies the purpose and scope of the ontology. Purpose answers the question “why is the ontology being built?” and scope answers the question “what are its intended uses and end-users?”;
- *conceptualization*, where one describes, in a conceptual model, the ontology that should be built so that it meets the specification found in the previous step;
- *formalization*, where one transforms the conceptual description into a formal model, that is, the description found in the previous step is written in a more formal form, although, not yet its final form;
- *implementation* where one implements the formalized ontology in a formal knowledge representation language;
- *maintenance*, where one updates and corrects the implemented ontology.

Besides the activities mentioned above that should be performed at each homonymous stage, there are other activities that can be performed during the whole life cycle, such as:

- *knowledge acquisition*, where one acquires knowledge about the domain either by using elicitation techniques on domain experts or by referring to relevant bibliography;

---

<sup>13</sup> We use the terminology proposed by METHONTOLOGY [Fernández et al. 1997] since it is the most consensual in the field.

- *documentation*, in which one reports in a document and along the implementation, what was done, how it was done and why it was done;
- *evaluation*, in which one technically judges the ontology.

Besides these activities there is also an activity that depends on the methodology:

- *reuse*, where one reuses other ontologies as much as possible. Most methodologies name this activity “integration”.

In Figure 9, we present a scheme of the activities involved in ontology development life cycle.

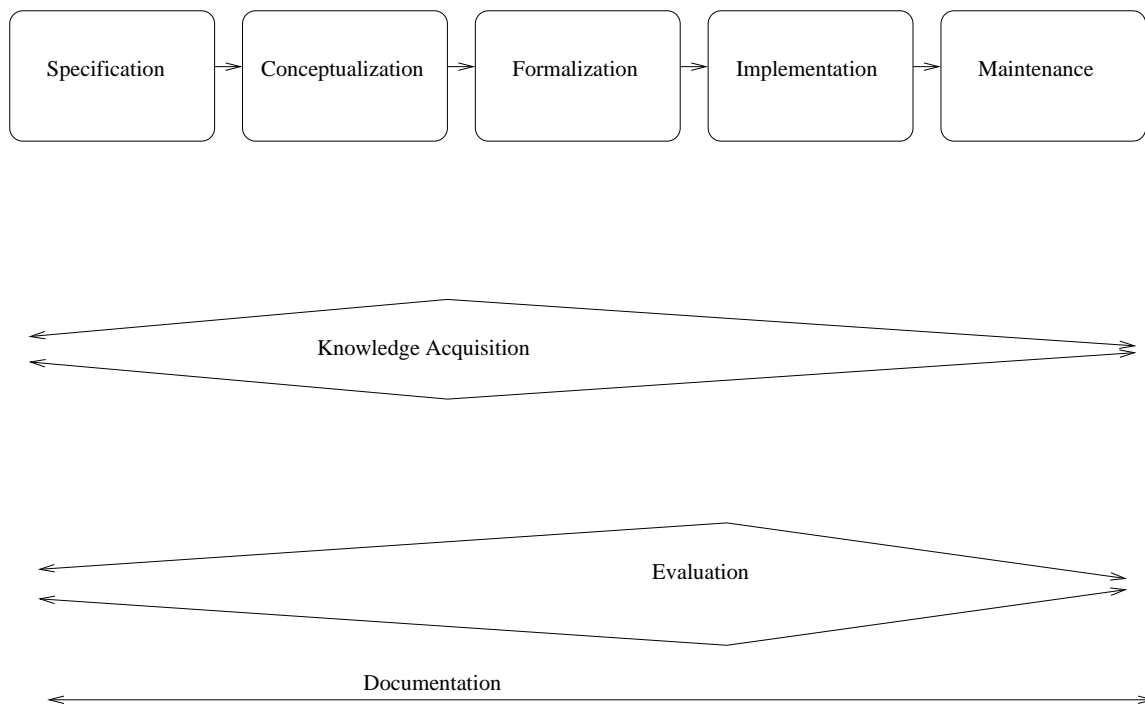


Figure 9: Activities in the ontology development life cycle

Although these activities have some influence from software engineering activities [IEEE-Std-1074-1995. 1996] they are different. The main differences are:

- in software engineering after requirement specification activities, design activities are not divided into conceptualization and formalization as most ontology methodologies propose,<sup>14</sup>

---

<sup>14</sup> The advantage of having formalization activities is to ease implementation and, in some cases, it even allows its automation.

- knowledge acquisition activities, (that take place during the whole life cycle), do not exist in software engineering processes and are an essential part of any ontology building methodology.

There are some activities that all of the most representative methodologies consider, namely, specification, conceptualization, implementation, evaluation and reuse.

In Figure 10 we compare waterfall [Royce 1970], iterative [Basili and Turner 1995] and evolving prototyping life cycle models,<sup>15</sup> enhancing how activities are scheduled along the life cycle and how the final product is developed.

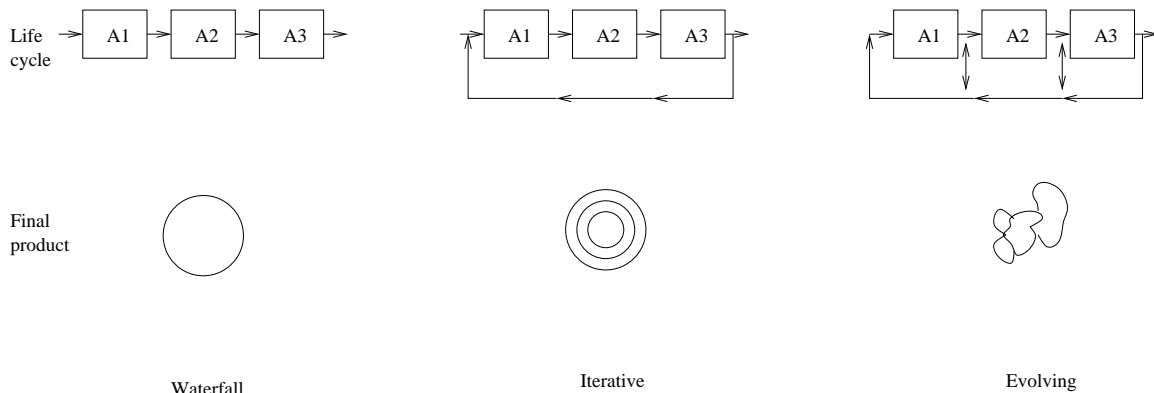


Figure 10: Comparison of life cycles

It is more or less *consensual* in the field that the development of an ontology follows an *evolving prototyping life cycle* rather than a waterfall or an iterative one. In an evolving prototyping life cycle, one can go back from any stage to any stage of the development process. As long as the ontology does not satisfy evaluation criteria and does not meet all requirements found during specification, the prototype is improved. One should note that evaluation should be performed along the whole life cycle, although some methodologies have a specific evaluation stage after implementation. Usually, one does not start over a new prototype in each “iteration”, it only improves the existing one.<sup>16</sup> Any part of the ontology that was identified as lacking quality or not meeting the desired requirements is improved. This kind of life cycle is different from a waterfall one, since it may go back to a previous stage of the life cycle. It is different from iterative life cycles since there is no a-priori planning of the several prototypes of the ontology that are going to be developed in each iteration of the ontology building process.<sup>17</sup>

<sup>15</sup> Also called evolutionary.

<sup>16</sup> Unless the existing prototype was found completely inadequate or the cost of modifying it outweighs the cost of building a new one.

<sup>17</sup> There is a fourth life cycle model called the spiral model [Boehm 1988]. In the spiral life cycle model there are no fixed phases and each spiral represents a phase where any

## Most representative methodologies to build from scratch

Although there is a set of ontology building methodologies proposals, none is widely accepted. Ontology building is still more of a craft than an engineering task. The main methodologies to build ontologies from scratch, are:

- TOVE methodology, which was used to build TOVE ontology (which is about enterprise modeling processes) [Fox 1995, Gruninger et al. 1995, Gruninger 1996],
- ENTERPRISE methodology, which was used to build the ENTERPRISE ontology (which is also about enterprise modeling processes) [Uschold et al. 1995, Uschold 1996, Uschold et al. 1996],
- METHONTOLOGY, which was used to build, among others, the Chemicals ontology (which is about the chemical elements of the periodic table) [Fernández et al. 1997, Fernández et al. 1999],

A comparative study of ontology building methodologies from scratch can be found in [Fernández 1999]. In the remainder of this section, we describe the most representative ontology building methodologies that address the problem of building ontologies from scratch. In the next section, we describe the methodologies that deal with reuse issues.

The TOVE methodology proposes the following stages:

- *capture motivating scenarios*, which are stories or examples that describe the motivation for the proposed ontology in terms of its intended applications,
- *formulate informal competency questions* based on the motivating scenarios. These are the questions that the ontology must be able to answer (these questions can be stratified, that is, the answer to a question can be used to answer more general questions),
- *specify the terminology* of the ontology within a formal language (it uses classical first-order logic, FOL),<sup>18</sup>
- *formulate formal competency questions in FOL* using the terminology defined in the previous stage,
- *specify axioms and definitions for the terms* in the ontology within the formal language,

---

life cycle model may be adopted (evolving, waterfall, etc.). In each spiral there are 4 steps: (1) determine objectives, alternatives, constraints; (2) evaluate alternatives; identify, resolve conflicts; (3) develop, verify next-level product; (4) plan next phase. The most important feature is the explicit consideration of risk. Ontology development does not follow a spiral life cycle since there are no explicit risk analysis activities identified so far in the field.

<sup>18</sup> Sometimes FOL is also called first-order predicate calculus, FOPC.

- *evaluate* the ontology by demonstrating the competency of the ontology with respect to the set of questions that arise from the applications that use the ontology,
- define the *conditions under which solutions to the questions are complete*.

The ENTERPRISE methodology proposes the following stages:

- *identify the purpose and scope* of the ontology,
- *build the ontology* by capturing knowledge,<sup>19</sup> coding knowledge<sup>20</sup> and reusing appropriate knowledge from existing ontologies,
- *evaluate* the ontology,
- *document* the ontology.

This methodology proposes a set of techniques, methods and guidelines for each stage. For instance, brainstorming and meetings with domain experts are suggested for knowledge acquisition. It proposes the use of a middle-out approach to produce the conceptual model of the ontology, instead of a bottom-up or top-down approaches. In a middle-out approach one begins by conceptualizing and defining the concepts that are more highly connected to other concepts since these are the most difficult to be correctly and accurately defined.

METHONTOLOGY proposes an evolving prototyping life cycle composed of:

- a series of development oriented activities that correspond to homonymous stages, such as:
  - *requirement specification*,
  - *conceptualization* of domain knowledge,
  - *formalization* of the conceptual model in a formal language,
  - *implementation* of the formal model,
  - *maintenance* of implemented ontologies;
- a series of support activities that are performed along the whole ontology building process, such as:
  - *knowledge acquisition*,
  - *documentation*,
  - *evaluation*,
  - *integration* of other ontologies;<sup>21 22</sup>

---

<sup>19</sup> Which includes: (1) identification of key concepts and relationships in the domain; (2) production of precise unambiguous text definitions for such concepts and relationships; (3) identification of terms to refer to such concepts and relationships.

<sup>20</sup> Explicit representation of the captured conceptualization in some formal language. This involves committing to some representation ontology, choosing a representation language and creating code.

- a series of project management activities, such as:
  - *planning*,
  - *control*.

Since we have used METHONTOLOGY terminology to describe the activities that compose the ontology building life cycle, the activities that compose METHONTOLOGY have already been described.

This methodology proposes a set of intermediate representations to help conceptualize knowledge [Gómez-Pérez et al. 1996] and a series of criteria to perform evaluation [Gómez-Pérez et al. 1995].

So far, none of these methodologies is mature enough or has a significant user community. Therefore, none has been accepted as standard. However, these methodologies are the most cited in the literature of the area. In <http://babage.dia.fi.upm.es/ontoweb/wpl/OntoRoadMap/index.html> one can find references to other ontology building methodologies.

## **Ontology reuse**

Although ontology building methodologies from scratch recognize reuse as part of the development process, none really addresses this issue. It is only recognized as a difficult problem to be solved.

There are two kinds of ontology reuse processes: merge and integration.

Merge is the process of building an ontology in one subject reusing two or more different ontologies on that subject [Pinto et al. 1999]. In a merge process source ontologies are unified into a single one, so it usually is difficult to identify regions in the resulting ontology that were taken from the merged ontologies and that were left more or less unchanged.<sup>23</sup> It should be stressed that in a merge process source ontologies are truly

---

<sup>21</sup> Earlier versions considered a reuse stage between formalization and implementation, but later versions considered integration as a support activity.

<sup>22</sup> Later versions [Amaya 1998] include a configuration management activity which records and maintains all different versions of documentation, software and ontology code.

<sup>23</sup> In some cases, knowledge from merged ontologies is homogenized and altered through the influence of one source ontology on another (in spite of the fact that source ontologies do influence knowledge represented in the resulting ontology). In other cases, knowledge from one particular source ontology is scattered and mingled with knowledge that comes from the other sources.

different ontologies and not simple revisions, improvements or variations of the same ontology.

Integration is the process of building an ontology in one subject reusing one or more ontologies in different subjects<sup>24</sup> [Pinto et al. 1999]. In an integration process source ontologies are aggregated, combined, assembled together, to form the resulting ontology, possibly after reused ontologies have suffered some changes, such as, extension, specialization or adaptation. In an integration process one can identify in the resulting ontology regions that were taken from the integrated ontologies. Knowledge in those regions was left more or less unchanged.

In Figure 11 we illustrate the differences between the two ontology reuse processes.

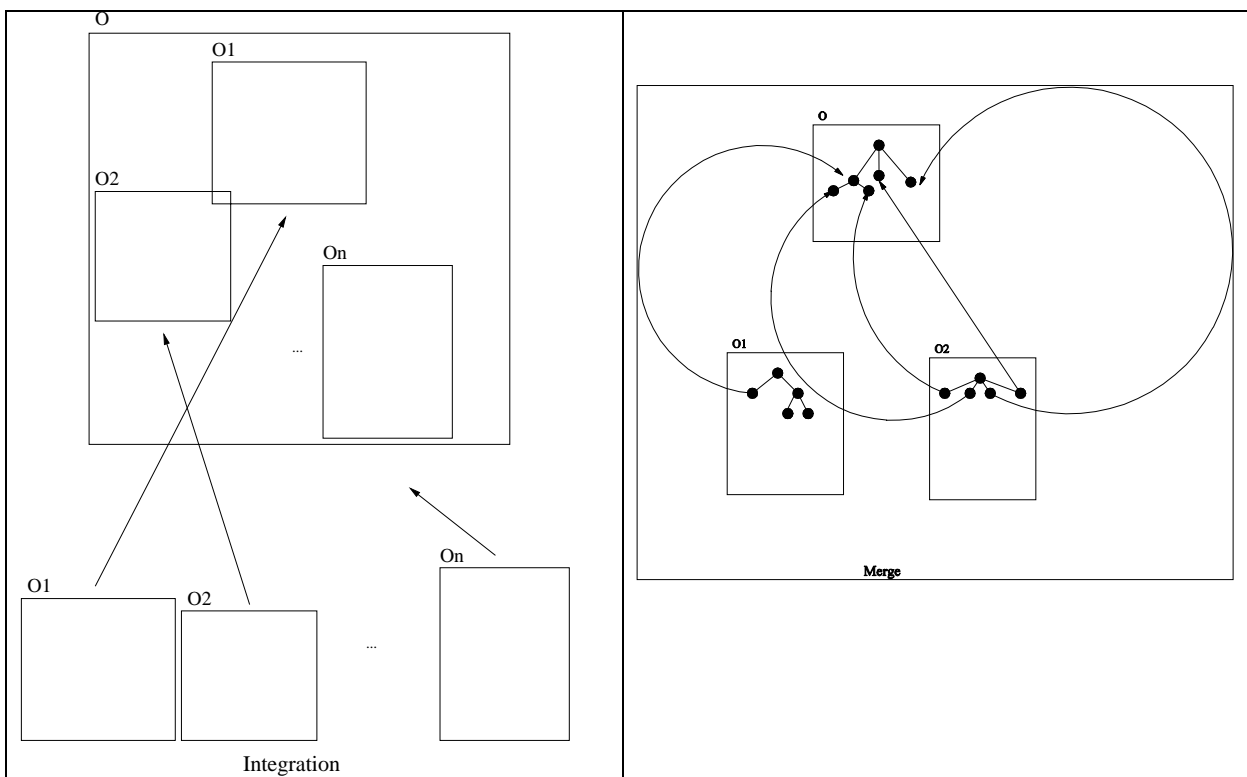


Figure 11: Comparison of reuse processes

It should be noted that both reuse processes are included in the overall process of ontology building.

<sup>24</sup> The subjects of the different ontologies may be related.

A lot of research work has been conducted under the merge area. There is a clear definition of the process [Sowa 2000], operations to perform merge have been proposed [Noy et al. 2000, Wiederhold 1994], a methodology is available [Gangemi et al. 1998] and several ontologies have been built by merge [Swartout et al. 1997, Gangemi et al. 1998]. The first tools to help in the merge process are now available [Noy et al. 2000, McGuinness et al. 2000].

In the integration area a similar effort is now beginning. The most representative ontology building methodologies [Uschold et al. 1995, Gruninger 1996, Fernández et al. 1999] recognize integration as part of the ontology development process, but none really addresses integration. Integration is only recognized as a difficult problem to be solved. They don't even agree on what integration is: for some it is an activity, for others a step. There have been some integration experiences [Arpirez-Vega et al. 2000, Pinto 1999, Pinto and Martins 2001b]. Recent studies [Pinto 1999, Pinto and Martins 2000] have shown that integration is a process of its own. Other important conclusions are that integration takes place along the entire life cycle and should begin as early as possible in the ontology building life cycle so that the overall ontology building process is simplified [[Pinto 1999, Pinto and Martins 2000]. The activities that compose the integration process are described in [Pinto and Martins 2001a]. Moreover, a methodology to perform this process has also been proposed [Pinto and Martins 2001c, Pinto and Martins 2001a].

Unfortunately, there still aren't many methodologies to help building ontologies by means of reuse. As far as we know, there is only one for each process: [Gangemi et al. 98] for merge and [Pinto and Martins 2001c] for integration. Therefore, nowadays this is still very much a research issue. However, these processes will be very important in building the AOS ontology. Therefore, one major contribution of this project, from the research point of view, will be a better understanding of reuse processes.

### **Ontology versioning**

An important issue that arises from the fact that ontologies may evolve is ontology versioning. Ontologies may change over time due to maintenance issues either because the domain they describe evolved (ex. new chemical elements are discovered), or the way we perceive that part of reality has changed (a new revolutionary scientific theory is proposed), or because errors were found in the ontology and had to be corrected. Although ontologies have been built since the early 1990, in the beginning in an ad-hoc fashion and from 1995 using methodologies, only now is ontology versioning issue arising as an important research issue. Existing ontology building frameworks (see part 6) do not deal with this issue. One of the important aspects of ontology versioning is that the tools maintaining the different versions of the same ontology should highlight the differences between versions. Important problems to be solved include how can an application built upon a given version of the ontology assure compability with newer versions of that ontology. Since the AOS ontology will most probably be built in a distributed fashion it will have to deal with the same problems that ontology building in the Semantic Web will face. An initial discussion of this problem can be found in [Klein and Fensel 2001].

## Methodology Example

Here we present a brief example of using a methodology to building an ontology. It shows but one approach that can be used to build a working ontology for a particular application in some domain.

**Purpose:** The purpose of the ontology in this example is to provide a basis for organizing a media collection in the crop-pest domain. Given a collection of about 1000 images of various crop pests, where each image includes a brief (one sentence) description, it is desired to build an ontology capable of describing the content of the images. The ontology will be used to assist people in locating images on a particular subject in the crop-pest domain.

**Raw Data:** A sample of the image captions is shown here. These will provide raw data for building the ontology. Notice how the text contains not only concepts (corn rootworm, peanut, leaf) but important relationships among concepts (that a particular pest occurs on a leaf, that a particular kind of damage is caused by a particular pest).

Southern Corn Rootworm Damage to Peanut Pod  
Wireworm Larva on Soil  
VBC on Stem - Dark Phase  
CEW on Leaf  
Large CEW Larvae on Leaf  
FAW Egg Mass on Leaf  
Small FAW Larva on Leaf  
Large FAW Larva on Leaf  
Adventitious Root Growth Caused By TCA Girdling  
Stink Bug Egg Mass in Leaf  
Hopper Burn - General View  
Red Necked Peanut Worm in Bud  
CEW on Leaf - Profile  
FAW Damage to Bud  
Hatching FAW Egg Mass

**Obtaining an Initial Vocabulary List:** Using software for extracting tokens and performing simple statistical techniques, a list of terms can be automatically extracted from the raw data. Here is a sample of the most frequent tokens appearing in the raw text:

1: on 217  
2: cotton 143  
3: soybean 91  
4: leaf 72  
5: photograph 69  
6: larva 60  
7: damage 57

8: armyworm 47  
9: peanut 47  
10: of 43  
11: bug 39  
12: boll 35  
13: stink 31  
14: beetle 31  
15: bollworm 28  
16: in 26  
17: adult 26  
18: southern 24  
19: feeding 22  
20: to 22

Furthermore, studying the frequency of bi-grams and tri-grams occurring in the raw text can help to identify words that occur together:

8: soybean leaf 22  
9: fall armyworm 21  
10: soybean photograph 20  
11: damage to 18  
12: peanut photograph 17  
13: bug on 16  
14: armyworm larva 15  
15: cotton boll 14  
16: damage on 13  
17: cotton bloom 13  
18: armyworm on 13

Though not all such word combinations are useful, do identify important patterns (soybean leaf, fall armyworm, armyworm larvae, cotton bloom).

The terms identified in this analysis also identify the concepts that should appear in the ontology. Remember to distinguish terms, which are expressions in a particular natural language, to the concepts that represent the meaning of the terms, ideally in a language-independent fashion.

**Identifying an Initial Taxonomy:** The next step is to take the list of concepts as described by the terms identified in the previous step, and form an initial class taxonomy. It is important to maintain consistence in the taxonomy, and this can be done by carefully identifying subsumption relationships. Figure 12 shows a portion of the initial taxonomy after adding just a few dozen concepts. Concepts were added one at a time, structuring the taxonomy as needed to accommodate each concept. New concepts were introduced (that were not part of the raw data) in order to describe categories of related concepts.

Already the top of the taxonomy, showing the most abstract concepts, is beginning to form.

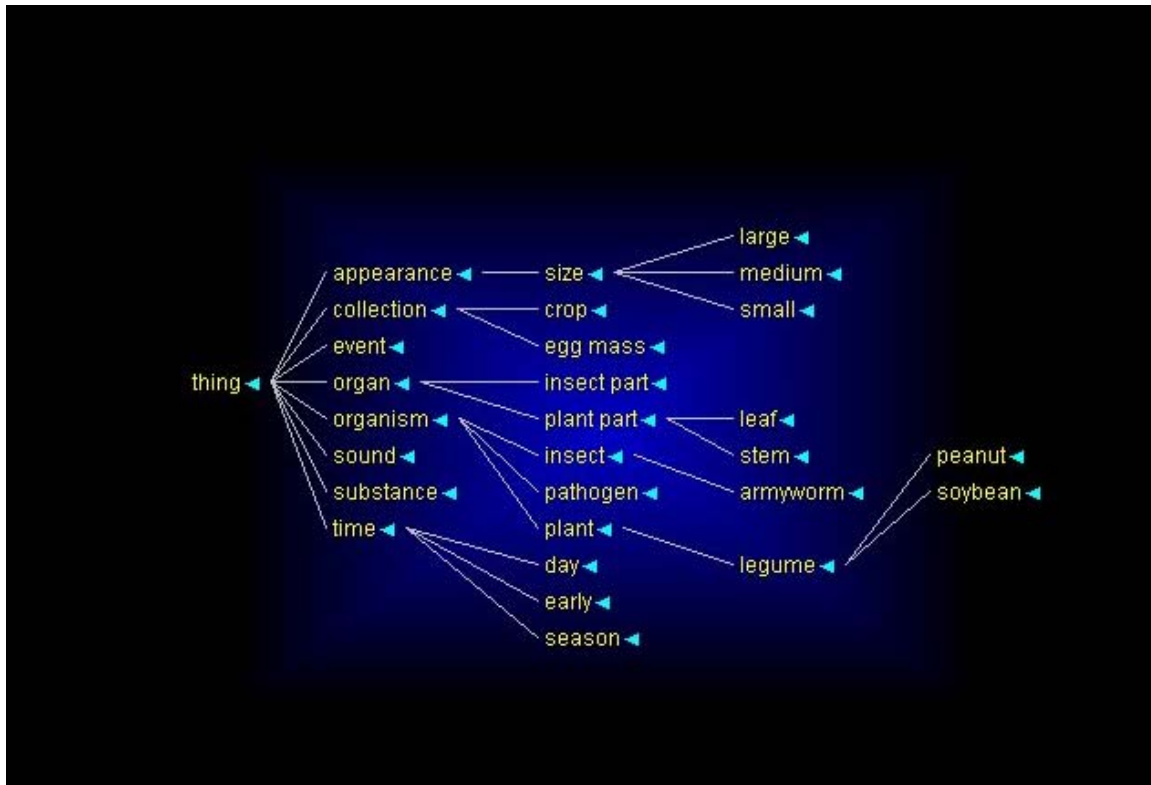


Figure 12. Initial Taxonomy for Crop-Pest Ontology

**Adding Restrictions and Axioms:** Concepts in the taxonomy can be further refined by adding attributes, attribute restrictions, and axioms expression relationships among concepts that must hold. For example, a “plant” can have attribute such as “pest” that must be a member of the “pest” category. A plant has parts which must be members of “plant part”. Events can have attributes describing agents and objects.

**Consistency Checking:** If subsumption relationships are described correctly, the attribute restrictions and axioms that are true of a class must also be true of any subclass of that class. Any additional restrictions imposed by a subclass must be consistent with restrictions inherited from superclasses. Consistency can be checked automatically.

**Incremental Modifications:** The ontology continues to grow as concepts are added. This is an on going process that continues even after the concepts identified from the raw data have been added.

**Evaluation:** At some point, the original goals for building the ontology will be satisfied. In this case, the ontology is adequate if the images from the media collection can be

cataloged by attaching them to concepts in the ontology. An evaluation can be done to see how well users can locate particular images by utilizing the ontology.

#### 4. Languages for Representing Ontologies

A knowledge representation language is used to build formal descriptions of concepts in an ontology. A knowledge representation language provides ways of building symbolic representations of what we know, along with formal methods for utilizing these representations to make inferences and draw conclusions. Somehow the meaning of concepts in the ontology must be represented in a way that can be manipulated by machines in order to perform the tasks of searching, discovering relationships among concepts, and looking for inconsistencies in the ontology.

How do we represent the meaning of a concept? To know a concept is, in a large part, to know the relationships between that concept and other concepts. In artificial intelligence, semantic networks are the knowledge representation method of choice in formally describing the relationships among concepts. For example, conceptual graphs [Sowa 1984] are a well known formalism for semantic networks created by John Sowa in the 1980's. To know the meaning of an expression such as "The black cat is on the mat" is to know how the individual concepts in the expression are interrelated. The graph in Figure 13 shows this relationship.

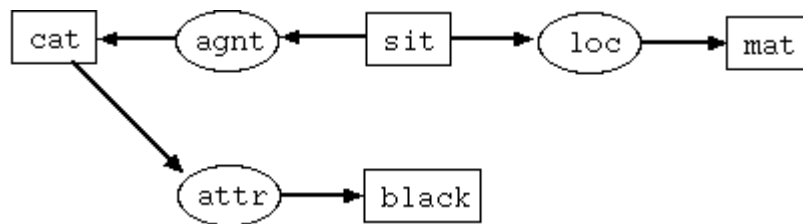


Figure 13. Conceptual Graph representation of "The black cat sits on the mat."

This section will describe formal methods for representing ontologies. We will present several languages, including several established or proposed standards such as Topic Maps, RDF, OIL, and DAML, to see what features they contain. All are based on semantic networks, and there are important, basic features that all these languages have in common. What makes these languages different is that some languages provide more features than others, that is, they differ in their level of expressiveness. As a result, they also differ in the kinds of inferences that can be applied, and the cost of applying them.

The implications for AOS include:

- Most existing languages for building ontologies share a set of common features.
- One can compare different languages with respect to this core set, as well as additional features introduced by a particular language.

- AOS should not attempt to design a new language, but rather draw on existing language and follow standards being created internationally.

## Description Logics

One family of semantic network formalisms in particular has emerged as an important influence in construction of ontologies, and in particular, is influencing recent XML-based standards for ontology languages (DAML+OIL). Going under the name “description logics”, the group originated with the KL-ONE language [Brachman and Schmolze 1985] developed in the 1980’s. KL-ONE has spawned numerous offsprings in the form of BACK, CLASSIC, CRACK, FLEX, K-REP, KRIS, LOOM, and YAK. The description logics created several important guidelines for evaluating semantic networks:

- The features of representational languages must be precisely and formally defined. Mathematical formulations of a language can be expressed using denotation semantics or model theoretic approaches, or other methods.
- The inferential operations that can be performed on systems constructed using these languages must also be well defined. The computational complexity of these operations must be specified.
- There is a practical limit to a language’s expressive power (the number and nature of features the language contains).

What kinds of inferential operations would we like to perform on an ontology? Here are a few:

- Automatic Classification/Subsumption. This is a fundamental operation that automatically determines if a given concept, A, belongs below another concept, B, in the taxonomy. It is the basis for not only providing assistance in building taxonomies, can be the basis for query processing [Beck et al. 1989].
- Concept Matching, Query Processing - Locate a concept based on a description. Find other objects in the ontology that are structurally similar to a particular object.
- Automatic clustering - Build new classes from a set of related instances by discovering how the instances are related.
- Consistency checking. Determines if an ontology that has been constructed is logically consistent (find any concepts that are placed together in relationships that are illogical).

Why not have a knowledge representation language containing all possible expressive features? Then it would approach the power of natural language, or formal logic. The reasons there must be a limit include:

- The main reason why we shouldn’t design complex languages, according to the formal studies done on computational complexity of description logics, is that

inferences performed on such systems become computationally intractable. In worst cases, this means that an answer to a question cannot be computed in any time frame. Or at best such an answer can't be computed quickly.

- Computational problems aside, the expense of building an ontology using extremely rich descriptions would be prohibitive.
- Training required of individuals building an ontology goes up as more advanced features are permitted in the language. The person building the ontology must understand how to properly use such features.

Thus we need a language that is expressive enough to build rich ontologies, but not so complex that it will be impossible to perform important computational operations, and that the ontology can be created without great expense, by people not requiring excessive training.

Fortunately, most of the languages being proposed for ontology construction on the Web meet these criteria, but largely because they are sensitive to the lessons learned from years of research on description logics.

### **Examples of Formal Languages for Building Ontologies**

It seems unlikely that the AOS project will invent a new representation language for ontologies, but rather would use, or at least borrow from, existing languages. There would also be a need to generate AOS representations in standard languages, and possibly more than one standard could be support. Several existing or emerging standard languages are described in this section.

These examples of languages for creating ontologies provide insight into the kinds of features that can be included in such languages. We present example in order to illustrate the range of possible language features. For a complete description for each language the reader is referred to the citations.

### **RDF**

RDF (Resource Description Framework) [W3C 2002] is a language built on top of XML for the purpose of describing Web resources. Figure 14 is an example RDF representation:

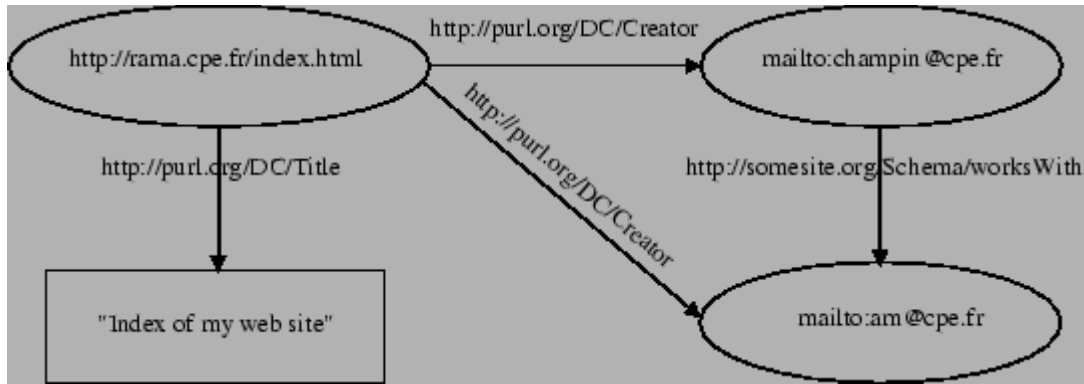


Figure 14. RDF Example [[RDF Tutorial Pierre-Antoine Champin](#)]

This figure says that a particular web site, referenced as “http://rama.cpe.fr/index.html” with title “Index of my web site” is created by two individuals, reference as “mailto:champin@cpe.fr” and “mailto:arn@cpe.fr.”

Some of the main features of the RDF language include:

**Resource:** A particular item of information, usual a Web site. Resources, represented as circles in Figure 14, are identified by URLs. This can be somewhat obscure as is apparent in Figure 14, but does serve as an unambiguous reference to a resource. It also shows that identifiers don’t always have to be expressions in English or other natural language.

**Class/Subclass:** Resources can be grouped into categories (Classes), and categories can have more specific categories below them (Subclasses) to form a taxonomy.

**Property:** Connects two related resources. For example “http://purl.org/DC/creator” connects the Web page with a creator. Properties are also URLs.

**Domain/Range:** The two resources participating in a Property can be restricted, a target “Domain”, and destination “Range” can be constrained to a particular subclass. For example, the Range of the “creator” property can be constrained to belong to class “Person”

**Containers:** Sets of resources can be grouped into containers. Containers can include “Set” (the elements of the container can appear in any order), “Sequence” the elements are sorted in some order, and “Alt” (the elements of the collection represent alternative choices).

## Topic Maps

Topic Maps [Pepper 2000] share a common goal with RDF of provide a way of indexing Web resources. Commercial tools such as Ontopia [] are available to create Topic Maps. The terminology used in Topic Maps is different, but means much the same thing as the

terminology used in other representation languages. The terminology of Topic Maps seems more natural and is easier to understand than the terminology used in RDF and other languages.

**Topic:** A generic category (a class), such as “Insect”, or “Crop”

**Topic Name:** A way of naming a topic. Naming is very important in ontology languages. Each object in the ontology must have a unique name so that it can be referred to unambiguously.

**Occurrence:** This is an instance of a topic. It corresponds to “Resource” in RDF, and loosely to objects and instances in other languages. An occurrence would be a particular Web site, such as a Web site about the topic “Insects”. Thus, Topics point to Occurrences. In both Topic Maps and RDF, the internal details of an Occurrence (or resource) are outside the scope of the representation language (they are arbitrary web pages). They are simply pointed to. In more advanced languages, objects are formally defined and are part of the representation language.

**Association:** A relationship between two topics. It is similar to a Property in RDF.

**Facets:** is like an association, but is a relation between a topic and a simple value. For example, the topic “Insect” might have a facet called “Number of Legs” which would be “6”. Facets can be considered a weak form of association, because the relationship is not between two objects (not between two Occurrences), but between an topic and a simple value.

**Scope:** Topic maps introduce the notion of Scope to handle a problem common to all ontology languages. It is desirable to partition an ontology so that objects related to a particular problem or subject are grouped together (in contrast to having all the objects together in one uniform, giant collection). Partitions also create contexts, such that the meanings of terms within a particular context are all grouped within the same scope. One problem this solves is that Topic Names having different meaning in different contexts can be partitioned into different scopes. So there can be one scope in which “Spider” means the arthropod with 8 legs, and another one in which “Spider” means a piece of software for searching the Internet.

## **OIL**

While RDF and Topic Maps can be considered light-weight languages for describing Web resources, DAML and OIL are languages being developed for building more complex ontologies within the framework of XML. OIL (Ontology Inference Layer) [Fensel et al. 2000] was created following the principles behind description logics. DAML (DARPA Agent Markup Language) can be considered an extension of OIL.

The main features of OIL include:

**Class:** A category of concepts, forming the familiar class/subclass taxonomy common in most semantic networks. Furthermore, in OIL a Class provides a formal definition of what it means to be a member of the class.

**Subclass:** A more specific category below a given class.

**Name:** Unique identifiers for classes and other elements

**Slot:** Used to describe properties of objects. Slots are associated with objects. They have a name and values (eg. a flower would have a slot for color and season).

**Slot Constraint:** Slot constraints place limits on the values a slot can have. For example, flower color would need to have a value that is a color, and flower season would need to have values that are times of the year. These constraints are located in classes and are used to build definitions of what it means to be a member of the class. All instances of a class must satisfy the slot constraints, and any subclass of a class must have slot restrictions that are logically consistent with the constraints of superclasses.

**Primitive/Defined:** Whenever possible, slot constraints specify both necessary and sufficient conditions for being a member of a class. Classes having such necessary and sufficient conditions are designated as being “Defined” classes. Often it is not possible to specify such conditions, and in many cases the constraints would be necessary (required) but not sufficient. Classes having necessary but not sufficient conditions are designated as “Primitive” classes.

Slot constraints and primitive/defined classes are key features of description logics. They provide the basis for automatic classification (determining whether one class is a special case of another, or whether an object belongs in a particular class) and other inferencing operations.

**And/Or/Not:** A slot can have more than one constraint, and they could be combined using boolean operations.

**Max-Cardinality:** A constraint can specify that the number of values in a slot must not exceed some maximum specified value.

**Min-Cardinality:** A constraint can specify that a slot must have at least some specified minimum number of values.

**Domain/Range:** These provide constraints on the concepts participating in a slot relationship. Restrictions are specified as a combination of classes such that the concepts in a domain (object having the slot) or range (concepts that are referred to as the property of the slot) must belong to these classes.

**Inverse:** Slots can be paired such that one slot is the inverse of the other. For example, if “damages” is a slot expressing a relationship from “pest” to “plant” (“pest” -> “damages”

-> “plant”), then “damaged-by” is an inverse slot for “damages”. (“plant” -> “damaged-by” -> “pest”).

Transitive/Symmetric: Relationships such as if A implies B and B implies C, then A implies C, or if A implies B and B implies A, used in associating slots with values.

## **DAML**

DAML, DARPA Agent Markup Language [2002], has many of the same features as OIL, plus some additional ones of interest:

Class: Same as for OIL

Class Identifier: unique object name, partitioned using name spaces

Subclass: Same as for OIL

Class Synonyms: Alternate names referring to the same class

SameClass/EquivalentTo: A way of saying that two classes are the same (they have the same members).

Complement/Union/Disjoint/Disjoint Union/Intersection: Set operations over the members of classes.

Individual/Instance: An object, an instance of a class.

Property: Relationship between two individuals. Corresponds to a slot in OIL.

Property Synonyms: Alternative names used to describe the same property.

Sub Property: A property that is a specialization of another property. This introduces the notion that properties can form a taxonomy.

Domain/Range: States that the individuals participating in a relationship must belong to certain classes.

Property Restriction: Corresponds to slot restrictions in OIL

Cardinality (exact, max, min) and Qualified cardinality range, unique: Restrictions on the number of values that a property can have.

Necessary & Sufficient Conditions: Similar to Primitive/Defined distinction in OIL.

Inverse/Transitive: One property can be the inverse of another (like in OIL). Two properties can be transitive. (Like in OIL).

Concrete domains (not yet implemented): Specific values for properties (string, integer, float, dollar, etc..)

Version: Version number of the ontology

Import: Include definitions from another ontology into the ontology currently being defined.

Thing/Nothing: All classes are subclasses of class “Thing” (it is the root of the taxonomy), no classes are subclasses of class “Nothing” (it is at the bottom of the class hierarchy).

## **Ontolingua**

Ontolingua (Gruber 1992) is a language based on KIF (Knowledge Interchange Format), and has been used to implement a number of ontologies, many of which are available at the Ontolingua server [<http://www.ksl-svc.stanford.edu:5915>]. Ontolingua contains a set of knowledge representation primitives including the following:

**Class:** A class is a representation for a conceptual grouping of similar concepts (essentially the same as a class in other languages). Each class includes a definition which specifies relationships that describe the class and the instances of that class. In addition a class can have logical statements called axioms that describe situations that cannot be represented using relationships and values.

**Subclass:** One class C is a subclass of parent class P if all of C’s instances are also instances of P. A class may have multiple superclasses and subclasses.

**SuperClass:** Superclass is the inverse of the subclass relation.

**Instances:** All of the terms in ontology that have an associated definition (i.e., classes, slots, relations, functions, and facets) are an instance of some class. Classes are instances of Class, functions are instances of Function, etc. An instance should not be confused with an Individual because an instance may be a class whereas an Individual cannot be a class.

**Individual:** A particular occurrence of a class.

**Relation:** A relation is any relationship among two or more concepts.

**Subrelation:** a subset of a relation. A relation and its subrelation have the same arity.

**Arity:** Arity is the number of arguments that a relation can take. If a relation can take an arbitrary number of arguments, its arity is undefined.

Slot: a relation between exactly two concepts.

Function: A function is a relation that relates one or more concepts to exactly one other concept. For example, mother is a function that relates an animal to exactly one female animal.

Domain: Restricts the concepts that can have a particular slot to being members of particular class.

Range: Restricts the values that a slot can have to being members of a particular class.

Exact-domain: More specific Domain

Exact-range: More specific Range

Cardinality: Restricts the number of values that a slot can have.

Inverse: A pairing of slots having reciprocal arguments. For example, if Vehicle has a slot "has-Wheel" then the Wheel class has an inverse slot of "Wheel-of". (like in OIL)

Axiom: An axiom is an expression in first order logic that must be true for classes associated with the axiom. Some relationships cannot be expressed using relations even with the available restrictions defined above. Axioms are used in such cases to express more complex relationships.

### **Advanced Features**

Additional language features can be conceived, although they are not part of any of the existing languages discussed so far:

Concrete Domains: It would be useful, especially for applications involving databases, if concepts could have relationships consisting of simple primitive values such as integer, string, real number, date, dollar amount, or binary data. DAML has proposed such concrete domains but this has not yet been implemented as part of the language. This would enable, for example, an object to have a digital image as a value of one of its properties, or a plant object could have properties with values such as "height" or "spread" and express these in real numbers.

Methods: Description logics are unique among object-oriented languages in that they do not contain methods. Methods impart behavior to an object. For example, a "plant" object could have a method to compute the stage of development of a plant based on temperature inputs (the plant could grow), giving rise to crop models and simulation. The problem is that methods are implemented directly in the code of some programming language, and what this code is doing is totally unclear to the inference engine (and often to humans who try to read the code). Thus they violate the requirement that the language

features be formally specified. A system containing methods cannot support basic inference operations such as those described earlier.

Rules: It would be relatively easy to include “IF-THEN-“ rules to the representation language. An example of a rule would be, “IF the temperature is below 2 degrees, and the dew point is below -5 degrees, THEN apply irrigation for cold protection”. This can be done in a clean and formal way. Most ontologies are not concerned with such reasoning, however it can be added as needed.

Lexical Information: Additional information needed to support natural language processing would turn the ontology into a lexicon. Features such as part-of-speech, grammar rules, and phrase patterns are discussed in section 6.

### **Core Set/Extended Set/Advanced Set**

All the language features above can be categorized into sets containing core, extended, and advanced features. It is proposed that an ontology modeling language for AOS consist of a core set of basic features, plus an extended set of more complex features, plus another extension of even more advanced features needed for special applications. Here is a hypothetical breakout of language features along these lines. This is intended as an example only, and is open for further discussion and development.

Core Set (RDF/Topic Maps):

- Class
- Object
- Object Identifier
- Superclass/Subclass/Parent
- Association
- Partitions

Extended Set (DAML/OIL):

- Facets
- Property Restrictions
- Primitive/Defined
- Domain/Range
- Subproperty
- Cardinality
- AND/OR/NOT
- Disjoint
- Collections

Advanced Set:

- Concrete Domains

Rules  
Methods  
Lexical Information

## 5. Ontology Tools

*In this chapter we refer and describe some of the most important ontology building tools currently available.*

There are a few ontology building tools available. A comparative study of ontological engineering tools can be found in [Duineveld 1999]. A few of these tools are freely available, such as:

- Ontolingua Server, available at <http://www.ksl-svc.stanford.edu:5915>
- Ontosaurus, available at <http://www.isi.edu/isd/ontosaurus.html>
- WebOnto, available at <http://webonto.open.ac.uk>
- PROTÉGÉ, available at <http://protege.stanford.edu>

There are a few features that all tools should have: a clear and consistent interface, provide help to the user by explaining the meaning of commands, be stable and quick. They should also have some sort of consistency checking and allow the reuse of ontologies kept in libraries (maintained by them). Ontology tools should also allow synchronous editing, ontology locking (and browsing by others when they are locked), some sort of change recognition, and export and import facilities. Moreover, these tools should allow the use of multiple inheritance. All these features are very important to build the AOS ontology, since this ontology will be built in a distributed way, involving multidisciplinary and multinational teams from rich and developing countries by all sorts of developers (from people mastering technology to naive users).

Most ontology building tools provide libraries of ontologies that can be reused through “inclusion” operations (the exception is PROTÉGÉ). However, before reusing these ontologies they should be evaluated and assessed, since most of them were not evaluated using the criteria accepted in the area [Gómez-Pérez 1995, Gómez-Pérez et al. 1995, Gómez-Pérez 1996, Gómez-Pérez 1999] before they were made available. Another important point to be stressed is the fact that most ontologies developed by private companies are not available in public repositories since they are regarded as company assets. Therefore, if in the AOS project one needs to reuse these ontologies these companies must be contacted if their ontologies or KOS are going to be reused and included in the AOS ontology. In <http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html> one can find references to existing ontologies.

The Ontolingua Server and Ontosaurus provide translators that can be used to provide first draft versions of ontologies in the desired languages. There are only a few translation attempts [Uschold et al. 1998, Russ et al. 1999]. In general, there are not many translators

available, their technology is still immature and improving existing translators is a rather difficult task. In [Uschold et al. 1998] the translation was done by hand and the conclusion was that this process is far from being a fully automatic process in the near future. In [Russ et al. 1999] the automatic translator between Ontolingua and LOOM available at the Ontolingua Server was used. Automatic translators were found still at draft level. Considerable human intervention was needed to improve automatic ontology translated versions. The conclusion was that, if translators are available, they should be used to produce initial versions that subsequently should be improved by hand. However, the translation process is, in general, complex.

The Ontolingua Server is the tool with the largest community of users. Ontologies are written in the Ontolingua language. The tool keeps a library of ontologies that can be reused to build other ontologies and one can add an ontology to the library and make it available to others. Different users all over the world can build an ontology collaboratively. The tool allows different users to work simultaneously on an ontology in a group session. Ontologies can be converted into different formats, either while importing or while exporting them. There is a good tutorial.

Ontosaurus is a Web browser for knowledge bases written in LOOM. There are a few ontologies available in its library. However, the tool is not very easy for naive users and an ontology can only be edited by one user at a time. Ontologies can be imported or exported in different languages.

Ontologies built with WebOnto are written in OCML, a graphical language. There are no exporting facilities, and one can only edit an ontology at a time (the ontology is locked for changes to other users, although they can view the changes).

PROTÉGÉ, has also a large community of users, but the tool is locally installed. It allows the creation of ontologies and it automatically generates the knowledge acquisition tool for entering the instances of the ontology. There are no cooperation or export facilities. It is very usable for a naive user but it does not have a library of ontologies. It also provides a graphical representation of the ontology.

In <http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html> one can find references to other ontology tools.

## **6. Advanced Topics – Databases, Natural Language Processing**

### **Database Management (Ontology as Database)**

There are several important relationships between ontologies and databases. Concepts in the ontology can map to specific data stored in a database, providing a term-oriented way to search the database. An ontology can map to many different databases, providing a way to integrate otherwise independent, federated databases. The ontology itself must be physically stored within a convenient device, and databases are also well suited to this

task because they can provide storage management, security, recovery, transaction management, and other useful services.

But there is another interesting relationship between ontologies and databases. As formal languages for building ontologies become more expressive, they become practical data modeling languages, and thus the ontology can also act directly as a database management system. The main extension required is that the ontology modeling language support instances (synonymous with objects, see section 4) and a rich set of data types. Objects correspond generally to records in a relational database and contain the actual data values. The ontology modeling language can support a rich set of data types, including primitives such as string, integer, real, or dollar, complex types including user-defined abstract data types (any class in the ontology is considered an abstract data type), multimedia type such as images and sounds, and thus provide the full power of conventional data modeling languages. Query languages over the ontology can be used to specify searches, and the semantic relationships expressed among objects in the ontology can be exploited by query processors to produce more content-oriented searches. A number of research systems have taken this approach, including CLASSIC [Borgida et al. 1998] and ConceptBase [Jeusfeld et al. 1998].

As an alternative to conventional relational database management systems (RDBMS), object database management systems (ODBMS) offer interesting features particularly suited for storing ontologies. Several commercial grade ODBMS are available, including ObjectStore [Object Design, Inc. 2002], Objectivity [2002], FastObjects [Poet Software, Inc. 2002], and Versant [2002]. The class-subclass, instances, and attribute-value structure of these databases closely parallels the structure of ontologies. Unfortunately these commercial ODBMS are not based on formal data modeling languages. They are persistent storage engines for C++, Java, or Smalltalk, allowing objects created using these programming languages to be stored in the database. Such programming languages do not make the best data modeling languages, and are not suitable for directly modeling ontologies, but can serve naturally as a storage layer (at a lower level) for physical storage of ontologies. On the other hand, the formal languages for building ontologies might offer a better theoretical basis for designing object databases.

Although conventional relational database management systems dominate the industry, the tabular structure of these databases is not directly compatible with the object-oriented nature of ontology structure. Nevertheless, several commercial products provide object-to-relational mappings [Duhl and Barry, 2001] so that designs can be constructed within an object-oriented framework, but physically mapped to a relational database for storage. Also, several leading relational database vendors including support object-extensions to the relational model, enabling complex data types (objects) to be stored as values of a relation. For instance, DB2 [IBM Corp. 2002] and Oracle [Oracle Corp, 2002] both support object extensions. These systems attempt to capitalize on the well developed commercial platform for relational data storage, while offering the richer modeling capabilities of object-oriented approaches. Whether this is more efficient than using a dedicated ODBMS for storage of ontologies is an open issue.

Query processing is an important function of any database management system, and there are also languages for expressing queries and other database operations known the data manipulation languages or DML. Structure Query Language (SQL) is the dominant query specification language used in industry. But SQL is not a natural way to describe objects and relationships needed for querying ontologies.

Consider this example query:

List books published by Addison-Wesley after 1991, including their year and title.

Stating this query in SQL would produce:

```
select title, year
from book
where 'Addison-Wesley'
and publication date > 1991
```

Here is the same query expressed in XQuery, the XML Query Language [<http://www.w3.org/TR/xmlquery-use-cases/>]:

```
<bib>
  {
    for $b in document("http://www.bn.com")/bib/book
    where $b/publisher = "Addison-Wesley" and $b/@year > 1991
    return
      <book year = "{$b/@year}>
        {$b/title}}
      </book>
  }
</bib>
```

Notice how the structure of the XQuery begins to resemble the structure of the desired objects (the <bib>...</bib> structure, plus the <book >...</book> structure).

Nevertheless, the from/where/result sections do not correspond with the structure of concepts in the ontology, resembling more the select/from/where statements in SQL.

Here is a hypothetical query language to get the same result, but utilizing a language that more closely resembles the ontology creation languages:

```
<book>
  <publisher>Addison-Wesley</publisher>
  <year><set><range><greater than>1991</greater than></range></set></year>
</book>
```

Here the query is itself like a class (the class of books published by Addison-Wesley after 1991). The query processing consists of automatically inserting (either temporarily or permanently) this “query class” at the correct position in the database, and in the process finding all instances that satisfy the restrictions specified by this class. The query

processor can exploit the semantic relationships expressed in the ontology in order to do this automatic classification. Then the resulting set of these instances is filtered by:

```
<book>  
  <title>?</title>  
  <year>?</year>  
</book>
```

to produce the desired format.

But these are approach that will require additional research and development before practical applications can be built. Here is a summary of areas to explore in the ontology-database relationship.

- Ways to map between an ontology and existing (conventional) databases
- Ontology modeling languages as database modeling languages
- Query Processing
- Transaction Processing
- Version Control
- Security/Integrity

### **Natural Language Processing**

Natural language processing (NLP) has many obviously useful applications, including allowing users to request information in natural language (natural language query specification), programs that read text and extract important information (information extraction), and converting documents written in one language into another (machine translation). Although there would be widespread interest in NLP should such facilities exist, NLP is an extremely difficult subject because of the enormous amount of supporting knowledge required. But ontologies, assuming they could be built, would provide an important component of this required knowledge, and an ontology should be built with support for NLP in mind.

Among other things, NLP requires an extensive dictionary, or “lexicon”, of the words that are understood by the NLP system. It is estimated that average individuals have an active vocabulary of around 20,000 words [Zechmeister et al. 1993], and a listening/reading vocabulary perhaps twice as large (40,000 words). Adding in terms from technical domains such as agriculture, the number is even larger (100,000 or more). However, it is not only the number of words known to competent speakers, but the extensive knowledge about each word, that is a limiting factor in NLP. Conventional dictionaries Webster’s Dictionary [Merriam-Webster, Inc. 2002], contain comparatively superficial definitions. Oxford English Dictionary [Oxford University Press, 2002] is far more extensive, but neither of these sources are in machine-readable form (the dictionary definitions are not in a form that can be manipulated by computers, even though they are in electronic form). An ontology promises to have both the volume of words, covering

the technical domain of agriculture, and the right kinds of formal relationships such that a machine can process the definitions for use in natural language understanding.

What follows is a brief illustration of the steps involved in interpreting natural language expressions, and some extensions beyond the ontology that would be needed to support NLP. We will use the natural language query “Find insect pests of soybeans that attack the leaves.”, as an example, showing how this sentence can be interpreted and mapped to database objects to get a specific answer using NLP techniques.

## Lexicon

The lexicon is the source for word information used in NLP. In addition to semantic relationships that are already part of the ontology, the lexicon would contain morphological and syntactic information not normally associated with the ontology. Thus, each word in the example query would appear in this lexicon, along with information such as part-of-speech, root forms (pest is root of pests), and associated phrase patterns.

## Phrase Patterns

Syntax, the physical ordering of words within a phrase or sentence, is vitally important for extracting semantic information, because syntax sets the stage for semantic analysis (determining meaning of a larger phrase by combining the constituent sub-phrases). Syntax can be modeled using “context-free grammars” (context-free simply means that the left side of the rule has a single term, and is thus independent of its immediate environment):

S -> NP,VP.  
NP -> Noun.  
NP -> Determiner, NP.  
NP -> Adjective, NP.  
NP-> NP,PP  
PP -> Preposition, NP.  
VP -> Verb,NP.

These rules say that a sentence (S), can be formed from a noun phrase (NP) followed by a verb phrase (VP), a noun phrase can be a noun (Noun), or a determiner followed by another Noun Phrase, or an adjective followed by a noun phrase, or a noun phrase followed by a prepositional phrase, and that a verb phrase can consist of a verb followed by a noun phrase. Such a context free grammar would only require a few hundred such rules to cover the English language. However, such grammars are highly ambiguous. Given that a single word can in the correct context be used in practically any part-of-speech, even a short phrase can be parsed many different ways using these grammars.

To counteract the ambiguity of context free grammars, phrase pattern grammars can be used that contain terms which are specific to the domain of discourse:

- 1 <health> and <health> <pest> <stage> with damage
- 2 <size> <pest> <stage> behind <plant part>
- 3 <pest> on <plant part>
- 4 <health> vrs <health> <plant growth stage>
- 5 <pest>-damaged <crop><plant growth stage>
- 6 <pest> on <crop> <plant part>
- 7 <pest><stage> on <crop><plant part>
- 8 dissected <plant part> showing <pest> <damage>

where words in < > represent categories that can be filled in by specific terms or other phrases patterns, and words by themselves are specific terms. These patterns would be used to match phrases such as:

- 1 Healthy and Parasitized BAW Larvae with Damage
- 2 Large BAW Larva Behind Bract of Bloom
- 3 Thrips on Cotyledon Leaf
- 4 Healthy vrs. Thrips Damaged Seedlings
- 5 Thrips-Damaged Cotton Seedling
- 6 TPB on cotton Bract
- 7 TPB Nymph on Cotton Bract
- 8 Dissected Square Showing TPB Feeding Damage

In addition, the phrases are now associated directly with lexical entries, for example, the second pattern would be tied directly to lexical entry for “behind”. In this case the phrases are more specific, and more associated with individual words (there is a direct connection between the phrase patterns and the lexicon). But the price to be paid is that tens of thousands of such patterns would be needed to cover a particular domain.

The sample query would require phrase patterns such as:

Find insect pests of soybeans that attack the leaves.

```
find <entity>
<organism> pest
<pest> of <crop>
<pest> that <behavior>
attack <plant part>
the <entity>
```

A parser using these patterns to analyze the query would produce this parse tree showing the syntactic relationships among the words in the phrase:

```
S
  Find
    <entity>
```

```

<pest>
  <pest>
    <organism>
      insect
    pest
  of
  <crop>
    soybean
that
  <behavior>
    attack
  <plant part>
    the
  <entity>
    leaf

```

### Parse Trees to Objects

The parse tree can be used to build semantic association among words in the query based on the established syntactic relationships. Here the existing relationships in the ontology are very important, as they are used to determine how words that are syntactically related are also semantically related. The conceptual structures in the ontology act as templates with slots to be filled from the words appearing in the parse trees. The ontology provides the critical link needed to produce semantic interpretations by providing a set of structures to choose from. An object representing the semantics of the query would look like:'

```

Entity
  pest:
    Name: ?
    Organism: insect
    Host: soybean
    Damages: leaf

```

### Object Matching

Once the natural language has been characterized as an object, it can be matched with other objects in the database. This determines where the new object belongs in the database. It can also be used to find similar objects in the database, and hence such matching essentially provides query processing. Matching the object representation of the query to instances in the database would locate pest names that would fill the position occupied by the question mark.

While this all works in theory, and there have in fact been several successful but experimental systems covering small domains based on this approach [Jacobs and Rao,

1990, Cowie et al. 1993], the main obstacle is to acquire the vast number of syntactic and semantic patterns required to cover a significant large domain such as agriculture. Construction of an ontology is certainly a valuable contribution to this effort.

### **Acknowledgments**

This part of the Background document is based on the introduction to ontologies presented in the PhD Thesis [Pinto 2000], in the article [Pinto and Martins 2001c], in the technical report [Pinto and Martins 2001d] and in the invited presentation to the AgentLink Special Interest Group on Agent Mediated Electronic Commerce [Pinto and Martins 2001e].

Section 2	[Pinto 2000]
Section 3	[Pinto 2000], [Pinto and Martins 2001c], [Pinto and Martins 2001d]
Section 6	[Pinto 2000] ,[ Pinto and Martins 2001e]

The AOS Launch Committee was helpful in guiding and reviewing this paper. Aldo Gangemi and Matthew Laurensen made specific contributions and suggestions related to several sections.

## References

- AGROVOC. 2002. AGROVOC Thesaurus. United Nations Food and Agricultural organization. <http://www.fao.org/agrovoc>.
- AltaVista. 2002. <http://www.altavista.com>
- Amaya, M. Dolores Rojas. 1998. Ontologia de Iones Mono-atómicos en Variables Fisicas del Medio Ambiente. Proyecto Fin de Carrera, Fac. de Informática, Unpublished Project Manuscript.
- Arpirez-Vega, J., A. Gomez-Perez, A. Lozano-Tello and H. Sofia Pinto. 2000. Reference Ontology and (ONTO)<sup>2</sup> Agent: the Ontology Yellow Pages. Knowledge and Information Systems. 2(4):387-412.
- Basili, V. and A. Turner. 1995. Iterative Enhancement, a practical technique for software development. IEEE Transactions on Software Engineering. 1(4).
- Beck, Howard W., Sunit Gala, and Sham B. Navathe. 1989. Classification as a query processing technique in the Candide semantic data model. Proceedings of the Fifth International Conference on Data Engineering. IEEE. Los Angeles, CA. pp. 572-581.
- Boehm, B. 1988. A spiral model of software development and enhancement. IEEE Computer. 21(5):61-72.
- Borgida, Alex, Ronald J. Brachman, Debora L. McGuinees, and Lori Alperin Resnick. 1989. CLASSIC: A Structural Data Model for Objects. Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data. Pp. 59--67.
- Borgo, Stefano, Nicola Guarino, and Claudio Masolo. 1996. Stratified Ontologies: The Case of Physical Objects. Proceedings of ECAI96's Workshop on Ontological Engineering. Pp. 5—16.
- Borst, Pim. 1997. Construction of Engineering Ontologies for Knowledge Sharing and Reuse. Ph.D. Dissertation. Tweente University.
- Brachman, R.J., Schmolze, J.G., 1985. An overview of the KL-ONE knowledge representation system. Cognitive Science. 9(2):171-216.
- CABI. 2002. CAB Thesaurus. CAB International. <http://www.cabi-publishing.org/Products/DBMANUAL/Thesaurus/Index.asp>.
- Cowie, Jim, Takahiro Wakao, Louise Guthrie, Wang Jin, James Pustejovsky, and Scott Waterman. 1993. Proceedings of the First Pacific Conference on Computational Linguistics. Vancouver. April 20, 1993.

DAML. 2002. DAML: DARPA Agent Markup Language Homepage.  
<http://www.daml.org>.

Duhl, Joshua, and Douglas K. Barry. 2001. Object Storage Fact Book: Object-Relational Mapping Release 5.0, July 2001. Barry & Associates. <http://www.object-relational.com/object-relational.html>.

Duineveld, A.J., R. Stoter, M. R. Weiden, B. Kenepa and V. R. Benjamins. 1999. Wondertools? A comparative study of ontological engineering tools. Proceedings of the Knowledge Acquisition Workshop, KAW99.

Fensel, D. L. I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. 2000. Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000), R. Dieng et al. (eds.), Lecture Notes in Artificial Intelligence, LNAI, Springer-Verlag.

Fernández, Mariano, Asunción Gómez-Pérez and N. Juristo. 1997. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Proceedings of AAAI97 Spring Symposium Series, Workshop on Ontological Engineering. pp. 33—40.

Fernández, Mariano, Asunción Gómez-Pérez, Alexandro Pazos Sierra and Juan Pazos Sierra. 1999. Building a Chemical Ontology Using METHONTOLOGY and the Ontology Design Environment. IEEE Expert (Intelligent Systems and Their Applications). 14(1):37-46.

Fellbaum, C. (Ed.), 1998. WordNet®: An Electronic Lexical Database, ISBN 0-262-06197-X. MIT Press, Cambridge, MA.

Fernández, Mariano. 1999. Overview of Methodologies for Building Ontologies. Proceedings of IJCAI99's Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends. pp. 4.1-4.13.

Fernández, Mariano. 1996. CHEMICALS: Ontología de Elementos Químicos. Proyecto Fin de Carrera, Fac. de Informática, Unpublished Project Manuscript.

Fox, Mark. 1995. SRKB Mailing List, 9th of June, 1995.

Gangemi, Aldo, Domenico M. Pisanelli and Geri Steve. 1998. Ontology Integration: Experiences with Medical Terminologies. Formal Ontology in Information Systems. IOS Press. Nicola Guarino. Ed. pp. 163-178.

Genesereth, Michael and Nils Nilsson. 1987. Logical Foundations of Artificial Intelligence. Morgan.

Gómez-Pérez, A., N. Juristo, and J. Pazos. 1995. Evaluation and Assessment of the Knowledge Sharing Technology. Towards Very Large Knowledge Bases. N.J.I. Mars. Ed. IOS Press. pp. 289-296.

Gómez-Pérez, Asunción, Mariano Fernández and António J. de Vicente. 1996. Towards a Method to Conceptualize Domain Ontologies. Proceedings of ECAI96's Workshop on Ontological Engineering. pp. 41—52.

Gómez-Pérez, Asunción. 1995. Criteria to Verify Knowledge Sharing Technology. Knowledge Systems Laboratory, Stanford University. Technical Report Number KSL-95-10.

Gómez-Pérez, Asunción. 1996. Towards a Framework to Verify Knowledge Sharing Technology. Expert Systems with Applications. 11(4):519-529.

Gómez-Pérez, Asunción. 1999. Evaluation of Taxonomic Knowledge in Ontologies and Knowledge Bases. Proceedings of the Knowledge Acquisition Workshop, KAW99.

Google. 2002. <http://www.google.com>

Gruber, Thomas R. 1992. Ontolingua: A mechanism to Support Portable Ontologies. version 3.0. Technical report, Knowledge Systems Laboratory, Stanford University, California.

Gruber, Thomas R. 1993. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition. 5:199-220.

Gruninger, Michael. 1996. Designing and Evaluating Generic Ontologies. Proceedings of ECAI96's Workshop on Ontological Engineering. pp. 53-64.

Guarino, Nicola and Pierdaniele Giaretta. 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. Towards Very Large Knowledge Bases. N.J.I. Mars. Ed. IOS Press. pp. 25—32.

Guarino, Nicola. 1997. Understanding, Building and Using Ontologies. International Journal of Human Computer Studies. 46(2-3):293-310.

Guarino, Nicola. 1998. Formal Ontology and Information Systems. Formal Ontology in Information Systems. Nicola Guarino. Ed. IOS Press. pp. 3-15.

IBM Corp. 2002. DB2. <http://www-3.ibm.com/software/data/db2>

IEEE-Std-1074-1995. 1996. IEEE Standard for Developing Software Life Cycle Processes. New York.

Jacobs, Paul S. and Lisa F. Rau. 1990. SCISOR: Extracting Information from On-line News. *Communications of the ACM*. 33(11):88-97

Jeusfeld, M.A. M. Jarke, H.W. Nissen, M. Staudt. 1998. ConceptBase - Managing Conceptual Models about Information Systems. In P. Bernus, K. Mertins, G. Schmidt. Eds. *Handbook on Architectures of Information Systems*, Springer-Verlag. ISBN 3-540-64453-9. pp. 265-285.

Klein, Michel and Dieter Fensel. 2001. Ontology versioning on the semantic web. *Proceedings of the Semantic Web Working Symposium*.

Lenat, Douglas and R. Guha. 1990. *Building Large Knowledge-Based Systems, Representation and Inference in the CYC Project*. Addison Wesley.

Lycos. 2002. <http://www.lycos.com>

Merriam-Webster, Inc. 2002. Merriam-Webster On-Line. <http://www.m-w.com/home.htm>

Mars, N.J.I. 1995. What is an ontology? The Impact of Ontologies on Reuse, Interoperability and Distributed Processing. *Unicom Seminars*. pp. 10-17.

McGuinness, Debora L., Richard Fikes, James Rice and Steve Wilder. 2000. An Environment for Merging and Testing Large Ontologies. *KR2000 Proceedings*. Anthony Cohn, Fausto Giunchiglia and Bart Selman. Eds. Morgan Kaufmann. pp. 483-493.

Michael Gruninger, Michael and Mark Fox. 1995. Methodology for the Design and Evaluation of Ontologies. *Proceedings of IJCAI95's Workshop on Basic Ontological Issues in Knowledge Sharing*.

Mizoguchi, Riichiro, Johan Vanwelkenhuysen and Mitsuru Ikeda. 1995. Task Ontology for Reuse of Problem Solving Knowledge. *Towards Very Large Knowledge Bases*. N.J.I. Mars. Ed. IOS Press. pp. 46-94.

NAL. 2002. NAL Agricultural Thesaurus. Natural Agricultural Library. United States Department of Agriculture. <http://agclass.nal.usda.gov/agt/agt.htm>

Neches, Robert, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator and William Swartout. 1991. Enabling Technology for Knowledge Sharing. *AI Magazine*. 12(3):37-56.

Noy, Natalya Fridman and Mark A. Musen. 2000. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. *AAAI2000 Proceedings*. AAAI Press. pp. 450-455.

Objectivity. 2002. Objectivity/DB. <http://www.objectivity.com>.

Object Design, Inc. 2002. ObjectStore. <http://www.odi.com>.

Oracle Corp. 2002. Oracle9i Database.  
<http://www.oracle.com/ip/dep/otn/database/oracle9i/>

Oxford University Press. 2002. Oxford English Dictionary. <http://www.oed.com>.

Patil, Ramesh, Richard Fikes, Peter Patel-Schneider, Don McKay, Tim Finin, Thomas Gruber and Robert Neches. 1992. The DARPA Knowledge Sharing Effort: Progress Report. KR92 Proceedings. B. Nebel, C. Rich and W. Swartout. Eds. Morgan Kaufmann. pp. 777-788.

Pepper, Steve. 2002. The TAO of Topic Maps. XML Europe 2000.  
<http://www.gca.org/papers/xmleurope2000/papers/s11-01.html>

Pinto, H. Sofia and J.P. Martins. 2000. Reusing Ontologies. Proceedings of AAAI 2000 Spring Symposium Series, Workshop on Bringing Knowledge to Business Processes, SS-00-03. AAAI Press. pp. 77-84.

Pinto, H. Sofia and J. P. Martins. 2001a. Ontology Integration: How to perform the Process. Proceedings of IJCAI2001's Workshop on Ontology and Information Sharing.

Pinto, H. Sofia and J. P. Martins. 2001b. Revising and extending the Units of Measure "subontology". Proceedings of IJCAI2001's Workshop on IEEE Standard Upper Ontology.

Pinto, H. Sofia and J. P. Martins. 2001c. A Methodology for Ontology Integration. Proceedings of the First International Conference on Knowledge Capture (K-CAP2001).

Pinto, H. Sofia and J. P. Martins. 2001d. Ontologies: How Can They Be Built? Grupo de Inteligência Artificial do Instituto Superior Técnico. Technical Report Number GIA 2001/04.

Pinto, H. Sofia and J. P. Martins. 2001e. Ontologies. Invited Presentation at the Agent Link meeting held in July, Prague, Czech Republic, to the Special Interest Group in Agent Mediated Electronic Commerce.

Pinto, H. Sofia, A. Gómez-Pérez and J. P. Martins. 1999. Some Issues on Ontology Integration. Proceedings of IJCAI99's Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends. pp. 7.1-7.12.

Pinto, H. Sofia. 1999. Towards Ontology Reuse. Proceedings of AAAI99's Workshop on Ontology Management. WS-99-13. AAAI Press. pp. 67-73.

Pinto, Helena Sofia. 2000. Ontology integration: Characterization of the Process and a Methodology to Perform it. Ph.D. Dissertation. Instituto Superior Técnico, Universidade Técnica de Lisboa.

Poet Software, Inc. 2002. FastObjects. <http://www.poet.com>

Royce, W. 1970. Managing the development of large software systems: concepts and techniques. Proceedings of IEEE WESTCON. pp. 1-9.

Russ, Thomas, Andre Valente, Robert MacGregor, and William Swartout. 1999. Practical Experiences in Trading Off Ontology Usability and Reusability. Proceedings of the Knowledge Acquisition Workshop, KAW99.

Salton, G. and M. McGill. 1983. Introduction to modern information retrieval. McGraw-Hill Book Company.

Sowa, John F. 1984. Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, Reading, MA.

Sowa, John. 2000. Knowledge Representation: logical, philosophical and computational foundations. Brooks/Cole.

Speel, Piet-Hein. 1995. Selecting Knowledge Representation Systems. Ph.D. Dissertation. Twente University.

Studer, Rudi, Richard Benjamins and Dieter Fensel. 1998. Knowledge Engineering: Principles and Methods. Data and Knowledge Engineering. 25(1-2):161-197.

Swartout, William, Robert Neches and Ramesh Patil. 1994. Knowledge Sharing: Prospects and Challenges. Knowledge Building and Knowledge Sharing. K. Fuchi and T. Yokoy. Eds. IOS Press. pp. 102-109.

Uschold, Mike and Martin King. 1995. Towards a Methodology for Building Ontologies. Proceedings of IJCAI95's Workshop on Basic Ontological Issues in Knowledge Sharing.

Uschold, Mike and Michael Gruninger. 1996. Ontologies: Principles, Methods and Applications. Knowledge Engineering Review. 11(2).

Uschold, Mike, Martin King, Stuart Moralee and Yannis Zorgios. 1998. The Enterprise Ontology. Artificial Intelligence Applications Institute, University of Edinburgh. Technical Report Number AIAI-TR-195. Note: To appear in the Knowledge Engineering Review, vol 13, Special Issue on Putting Ontologies to Use. Mike Uschold and Austin Tate. Eds.

Uschold, Mike, Mike Healy, Keith Williamson, Peter Clark and Steven Woods. 1998. *Ontology Reuse and Application. Formal Ontology in Information Systems*. Nicola Guarino. Ed. IOS Press. pp. 179-192.

Uschold, Mike. 1996. *Converting an Informal Ontology into Ontolingua: some experiences*. Proceedings of ECAI96's Workshop on Ontological Engineering. pp. 89-99.

van der Vet, Paul E. Ed. 1996. *Proceedings of ECAI96's Workshop on Ontological Engineering*. ECAI96.

van Heist, G, A. Th. Schreiber and B. J. Wielinga. 1997. *Using Explicit Ontologies in KBS Development*. International Journal of Human-Computer Studies. 46(2-3):183-292.

Versant Corp. 2002. Versant. <http://www.versant.com>

Vossen, Piek. 2001. EuroWordNet. <http://www.hum.uva.nl/~ewn>

W3C. 2001. *Semantic Web*. World Wide Web Consortium. <http://www.w3.org/2001/sw>.

W3C. 2002. *Resource Description Framework (RDF)*. World Wide Web Consortium. <http://www.w3.org/RDF>.

Weiss, S.M., C. A. Kulikowsky, S. Amarel, A. Safir. 1984. *A model-based method for computer-aided medical decision making*. Readings in Medical Artificial Intelligence, the First Decade. Addison Wesley.

Wiederhold, Gio. 1994. *Interoperation, Mediation and Ontologies*. Proceedings of the International Symposium on the Fifth Generation Computer Systems, Workshop on Heterogeneous Cooperative Knowledge-Bases. W3:33-48.

Zechmeister, E.B., D'Anna, C., Hall, J.W., Paus, C.H. & Smith, J.A. 1993. *Metacognitive and other knowledge about the mental lexicon: Do we know how many words we know?* Applied Linguistics 14(2):188-206.