Combining Global and Local Ontology handling in a Multiagent System

Ramon F. Brena and **Hector G. Ceballos**
Tecnologico de Monterrey, Mexico
{rbrena, ceballos}@itesm.mx

Abstract

Ontologic knowledge is getting more and more important in agent-based systems, and its handling is becoming crucial for successful applications. But placing all the ontology-handling capabilities in each of the system's agents could make them too heavy. We propose a combination of local and global ontology handling, where part of the ontology is handled locally, using a "client component", and the rest of the ontological knowledge is handled by an "ontology agent", which is accessed by the other agents in the system through their client component. We propose specific methods for representing, storing, querying and translating ontologies for effective use in the context of the "JITIK" system, which is a multiagent system for knowledge and information distribution. We report a working prototype implementing our proposal.

Introduction

It is widely accepted that communication is an absolute requirement for many multiagent system applications. This requires, of course, low level facilities for physical connectivity as well as higher level internet protocols and even inter-agent communication protocols. Even if these are not completely solved problems, what is right now most challenging is taking into account the *meaning* of agent messages. But this is one crucial aspect that we have to deal with in order to build realistic open agent-based applications (Nwana and Ndumu 1999).

The term *ontology* refers to a definition of meanings for terms used in inter-agent communications (Wooldridge 2002). Ontologies allow to define concepts and their relations, properties, operations, and the like in a structured fashion. Open standards like DAML- OIL(Conolly et al. 2001), or more recently OWL(Dean et al. 2003) allow to publish ontologic knowledge in a way understandable both by humans and machines.

Even if a representation standard is set, it remains to be decided where to put each piece of knowledge to be represented. Some efforts like the Cyc project (Lenat 1993) suggest to build huge centralized repositories of encyclopedic knowledge. Others considered this impractical in terms of performance and robustness, and prefer decentralized approaches (Weichhardt 2002). But handling distributed ontologies generates new difficult problems as well, namely: 1) Where to put or get pieces of knowledge -i.e. how to distribute the knowledge; 2) How to maintain some degree of coherence among the different pieces -or even versions- of ontological knowledge. Further, independent partial ontology repositories could evolve independently and diverge.

The method we will present in this paper is intended to be used in the context of the JITIK project (Brena et al. 2001). JITIK -which stands for "Just-In-Time Information and Knowledge"- is a multiagent-based system for disseminating pieces of knowledge among the members of a large or distributed organization, thus supporting a Knowledge-management (Liebowitz and Wilcox 1997) function. Although our ontology-handling proposal was primarily intended for its application in the JITIK system, which imposed some architectural as well as other restrictions, our proposal is applicable in principle to a wide range of agent-based systems.

Our approach

In this paper we propose a solution to one of the aspects of this situation: we propose a method for combining centralized with distributed ontologies. We consider a central repository encapsulated in an "ontology agent", (OA) providing answers to questions about the ontology to the other agents in the system. But besides that, we endow each agent in the system with a "client ontology component" (COC) which gives it basic ontology handling capabilities. This arrangement works in the following way:

- Standard agents start with a subset of a common ontology, which is loaded at startup from an internet resource.
- Standard agents use their local ontologies, handled by the COC, as long as the local knowledge suffices for the agent's activity.
- When further knowledge is required -for instance, an unrecognized term arrives from other agent- the COC queries the OA, and receives a taylored addition to the basic ontology, that allow the agent to continue working. The COC stores locally the ontology addition so it could be used in the future.

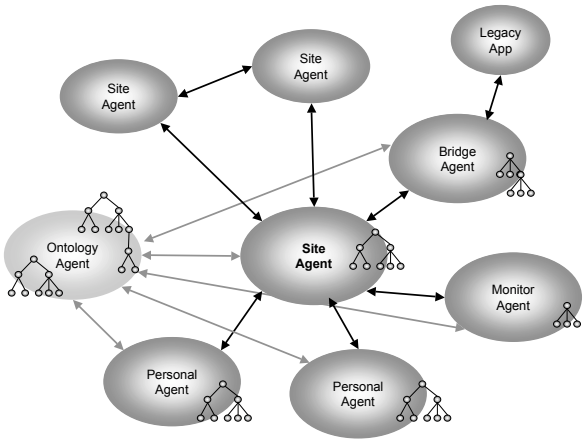

Figure 1: JITIK agents

This solution simplifies some of the inherent complexities of knowledge distribution, because:

1. There is no risk of incoherence -every piece of knowledge comes ultimately from the common ontology -either from the initial ontology or as a result of a query to the OA.
2. There is no *a priori* decision about how to distribute or specialize the knowledge, because local knowledge gets specialized automatically as agents' operation proceed.

In the next sections we detail our method, the prototype and the experimental results. Finally we draw some conclusions.

The proposed solution for ontology handling in JITIK

In Figure 1 we depict JITIK's architecture, composed of several kinds of agents, like the *Site agent*, taking in charge the distribution of information to several *personal agents*, which interact with an end user; there are as well *bridge agents* for interacting with traditional software (*legacy applications*). Site agents are the heart of a "cluster" composed by one site agent and several personal agents served by the former. In an organization, clusters would be associated to departments, divisions, etc., depending on the size of them. Networks can be made up connecting several site agents. Distributed organizations like multinational companies would have a web of many connected site agents. There are also *ontology agents*, which we will discuss in the following.

From the point of view of ontology handling, we will classify the agents in two categories: *ontology agents*, and "*regular*" agents, which are all the other agents. Sometimes we will call "client" agents the regular agents, because they act like a client with respect to the ontology agent. The OA and the clients correspond to the "initiator" and "respondent" roles in standard agent design methodologies like (Collis and Ndumu 1999).

Client agents try to fulfill their knowledge needs using the knowledge in the COC. If necessary, the COC makes a query to the OA, and interprets and use the answer, and eventually incorporates it to the local knowledge.

Regular agents access ontologies through the COC introduced before, and the COC eventually communicates with the OA. The architecture of both the ontology agent and a client agent is depicted in Figure 2. It is shown there OA at the left side, with some internal components to be discussed later, and a client agent at the right side.

Let us first take a look to the OA.

The OA encapsulates the functionality for playing the role of a knowledge provider, storing the ontology conveniently encoded, translating, interpreting and executing incoming queries, then translating back the results to a format understandable by the client agents. Translation is sometimes necessary because the encoding for storing knowledge and answering queries, which is mandated by performance requirements, could be different from the one used in the client agents. This format separation provides a layer of independence, so that the ontology representation could be changed in the OA without impact to the client agents.

Client agents access ontology definitions through their COC. At startup they load a *base ontology*, which is evidently application dependent, and try to use it as long as it suffices for agent's work. In the JADE system, ontologies are needed for message validation purposes in the first place. Every term in agents conversations should be validated against a definition in an ontology. Thus, normally the base ontology will contain definitions of common terms. The size of the base ontology is a tradeoff between space efficiency -asking for a small initial ontology- and time efficiency -asking to maximize the coverage of the local knowledge so remote queries are minimized.

Aiming to decentralize access to the ontology, we propose a combination of global and local ontology handling, where the agents clients can access part of the ontology locally, or remotely, asking directly to the OA. Local access is going to be encapsulated in the COC which is attached to the client agents.

At agent's startup, the COC is responsible for fetching -normally from an internet location- a base ontology. This mechanism is general enough to be customized so that different types of agents load different base ontologies, though we have not done this in our prototype.

In order to overcome the limitations of the base ontology, the COC is responsible for accessing the OA for extending its ontology knowledge, through the query mechanism we have been describing. The results of a query are incorporated by the COC to the local ontology, thus extending automatically the ontology as needed.

In this model, the very existence of the OA is transparent to the client agent, as it directs every query to the COC, this one takes in charge the whole process until an answer arrives to the agent -either from a local COC consultation or from a query from the COC to the OA.

As we can see in the diagram of figure 2, the COC has the following elements:

- *Local Ontology representation*. It allows to store a subset of the ontology, and supports local querying.

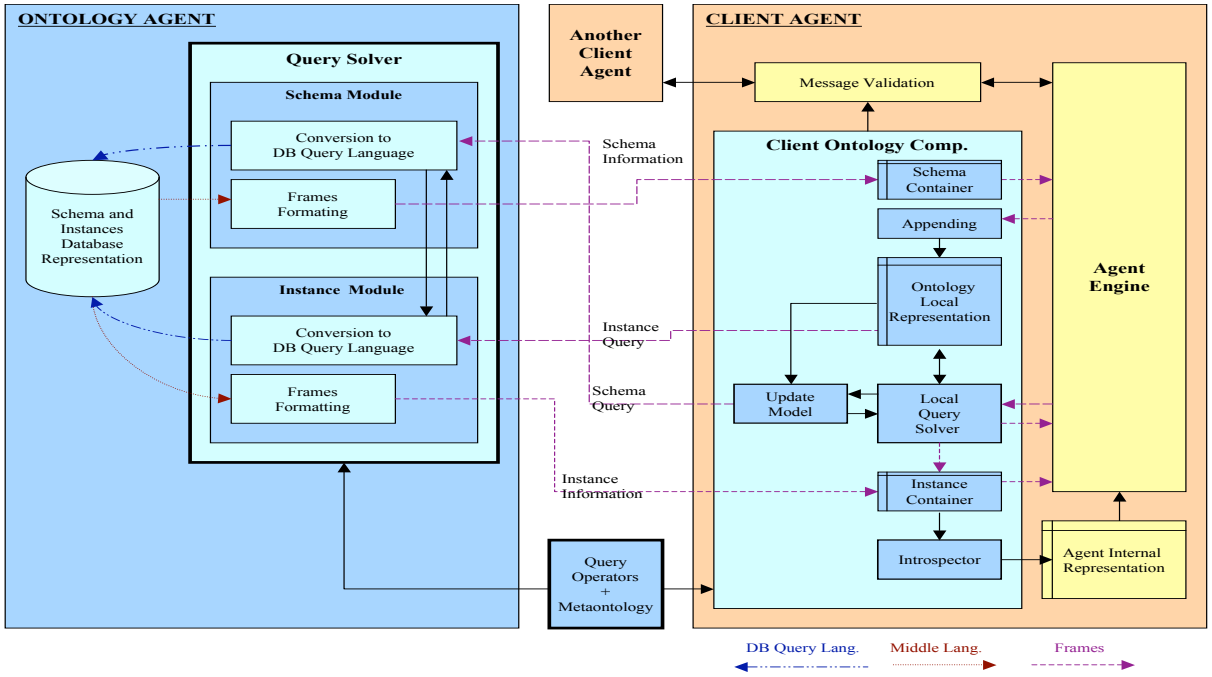

Figure 2: Ontology handling architecture

- *Local query solver*. Interface between the agent itself and the ontology view. Exposes methods usable by the agent to query about the ontology schema or instances.
- *Message validation*. As the COC contains definition of terms from the base ontology and from queries to the OA, it allows to validate messages in terms of an ontology, as it is required by the JADE platform.
- *Schema container and Instance container*. We wanted schema information to be kept separate from instance information for performance reasons, particularly when a large number of instances is involved. Instance information can be accessed either directly from the client agent or exporting a Java class through the use of the *Introspector*.
- *Appending mechanism ("update model")*. New knowledge coming from OA as a response to a query is incorporated to the local view. Of course, imprudent use of this facility could bloat the COC.
- *Introspector*. Converts frame representations to agent internal representations objects (Caire 2002).

Ontology representation

In the preceding discussion we were assuming an adequate encoding for the ontologies was available. Now we discuss this issue.

We analyzed existing ontology encodings, requiring that they should have enough descriptive power, and at the same time be widespread used among the scientific community. The descriptive power we consider as satisfactory is that of Description Logics (Baader et al. 2003), which is broadly implemented in standards like OWL (Dean et al. 2003).

Next we had to find an adequate mechanism for querying the ontology agent from client agents. This includes query formation, query execution, and response interpretation.

We wanted to be able to ask questions not only about specific instances in the ontology, but also about the concepts and their relations themselves. The need to make inferences on the ontology entails the need to define the minimal elements of the ontology. These elements are specified in a *metaontology* that will serve as a basis to encode the ontology, as well as the questions to the ontology and the fragments of it that conform the answers. In the metaontology we consider a set of concepts like *Class*, *Property* and others, and each concept has *attributes*.

We examined several ontology formats created for Semantic Web (Berners-Lee et al. 2001), as they are the best known today. In particular, RDF (Lassila and Swick 1999), DAML+OIL (Conolly et al. 2001), and recently OWL (Dean et al. 2003), contain the elements we defined in the metaontology, and are expressive enough so we can make inferences on them. We decided to use DAML+OIL, as it provides good expressive power for our purposes, and has reached widespread acceptance. There is no point in describing in detail the ontology formats, as it is just standard DAML+OIL. A fragment of our example ontology looks like the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
...
<daml:Class rdf:about="itesm.daml#ResAssist">
  <rdfs:label>ResAssist</rdfs:label>
  <rdfs:subClassOf>
    <daml:Class rdf:about="itesm.daml#Assist"/>
...
<daml:ObjectProperty
    rdf:about="nsJITIK/itesm.daml#head">
  <rdfs:label>head</rdfs:label>
  <rdfs:domain>
    <daml:Class rdf:about=
      "itesm.daml#OrganizationalUnit"/>
  </rdfs:domain>
...
```

Prototype

Since in the JITIK project we program using the JADE multiagent development platform (Bellfemine 2002), we considered ontology handling support recently incorporated to JADE.

From version 2.5, JADE incorporates some support for ontology handling. Using these facilities we built the COC that gives to the client agents immediate access to the local part of the ontology.

To do this, it was necessary to redefine the "Ontology" class, which encapsulates the ontology definition, as well as to implement access methods for consulting it.

In JADE, query operators can be defined using *predicates* (PredicateSchema) and *quantifiers* (AbsIRE). The metaontology is defined in terms of *concepts* (AbsConcept class) and *aggregates* (AbsAggregate).

Another JADE facility is to use the *Introspector* class, which allows to translate between Java objects and text-encoded objects ready for being sent in agent messages, in *frames* format.

We have developed so far a somewhat simplified implementation of the ideas presented above. The simplifications we introduced are the following:

- The Jena Toolkit (Jena) we decided to use for the prototype has incorporated a persistent storage facility, which was easier to use than a separate database-style storage for the ontologies, as was suggested in the conceptual diagram (figure 2).
- Access to ontologies on the client agents and on the OA are identical, both based on a *ClientOntology* class we developed, which calls Jena package facilities. So ClientOntology is implemented in both the COC and the OA.
- The COC does not redirect queries to the OA. The client agent instead has to know which component to query, either the COC or the OA.
- No distinction is made between instances and schema for storing purposes.
- RDF multitype instances are treated as *Compound Class*' instances. This compound classes are created at runtime when a *mixed instance* arrives(Ceballos 2003).

Query solving

Queries consist of the following:

- A quantifier, which indicates if all the results are needed, or we want to check if there are items with a given description.
- A variable, where the result data type is specified.
- A query operator.

Query operators are defined so that their evaluation is made in two steps: first, the characteristics of the objects are specified, and second, the element of the found objects is indicated. During the first step, Jena extracts a list of DAML+OIL schemas satisfying the given specification, and in the second step results are constructed.

For instance, assume that we want to know which properties are defined in the class "Worker". We will use the ALL quantifier, so the properties themselves, and not just their number, are returned. Now we define a variable "x" of type *CLASS_PROPS*, which can store a list of properties defined in a class.

Finally, the DescWhere operator is introduced, using as parameters a filter and the results structured. In the example below the filter is a class name (Worker), and the result structure uses the result variable "x" to store answers.

The query in our example would be as follows (ommited details appear as "..."):

```
(ALL
  :VARIABLE ... :NAME x)
  :PROPOSITION (DESCWHERE
  ...
    :CLASS_NAME Worker))
```

Using our example ontology, the following query result indicates that Worker class' properties are *id*, *email*, *interests* and *name*:

```
(RESULTS :RESULTS_SET ... (PROPLIST
      #1 (... :PROP_NAME id)
      #2 (... :PROP_NAME email)
      #3 (... :PROP_NAME interests)
      #4 (... :PROP_NAME name)
```

It shoud be noticed that the returned property list in this example includes not only the direct properties of Worker, but those defined in its superclasses as well.

In the prototype we achieve good COC-OA integration, as the query results are sent to the COC, which forwards them to the agent, and in addition incoporates those results to the local ontology. We are taking advantage of Jena's mechanism for merging ontologies. When a query response arrives from the OA, instead of arriving directly to the client agent it passes though the COC, allowing it to incorporate those results as an extension to the base ontology.

Experiments and Results

We designed and carried out experiments aiming to ensure that every possible query could be solved by our system, and that translations work properly. What we intended to prove refers to the correctness of our methods –as well as

our implementations–, but we proved nothing yet about performance issues (see section). We assumed, of course, that the software we are building upon (JADE, Jena) works correctly.

We carried out a formal testing methodology, sorting first all the possible queries in a linear sequence, and then taking randomly some of the queries, until a sample size is met. Details of our testing method are reported in (Ceballos 2003).

We used a test ontology about our university (Monterrey Tech), representing the organizational structure, as well as properties of people studying and working there. The DAML files (itesmcore.daml and personal.daml, for schema and instances), are accessible by internet. The tool OilEd (Bechhofer et al. 2001) was used to edit our test ontology. Some manual adjustments were necessary to make it usable by Jena.

The main result from our experiments was that all of the sample queries were correctly answered. Technical details of our proof methodology are available in a separate report (Ceballos 2003).

It remains to be proved that our approach outperforms both completely centralized and completely distributed approaches, when we consider various time and space tradeoffs (see conclusions).

Related work

Although there are several toolkits for ontology handling (Alexaki et al. 2002; Ontoprise), some with inference mechanisms, none of these are really suited to multiagent systems. Thus their adaptation require an important effort. Further, almost none include mechanisms specificallyl designed for distributing ontologies in a flexible way.

In the KAON project (Maedche et al. 2003) the emphasis is in reusing ontologies and propagating changes in distributed ontologies. It handles a registry with known ontologies URIs in several "Ontology Servers", similar to out OA, which take in charge ontology loading, updating and propagation. Each Ontology Server provides a querying service for its agent community. In the Ontology Server a local copy of the original RDF ontology is stored, taken initially from an URI. Our OA, on the other hand, uses full DAML+OIL ontologies. In our approach, the COC is responsible for updating the ontology, and the copy at the OA remains static. On the "minus" side, we have not taken into account ontology evolution, though we plan to change this in the future (see section).

In the COMMA project (Gandon 2003), as in JITIK, there is a global ontology, which is propagated to known agents. Each agent receives a complete copy of the ontology, and is able to solve queries about it. COMMA uses RDF for ontology coding, and includes an API for ontology access by agents. Obviously this approach lies in the centralized extreme of the spectrum.

In the FRODO system (Van Elst and Abecker 2002) there are specific *roles* with respect to ontologies: the "ontology provider" and the "ontology consumer". Providers take in charge ontology services, including updating. Consumers just use ontologies to run their applications. In FRODO there are two ontology distribution levels, with respect to a given agent community: internal and external or "intersystem". There are in FRODO three ontology handling categories: ontology *use*, ontology *evolution* and ontology *socialization*. Emphasis is given to query formulation and solution. In JITIK we consider use and ontology socialization, but not (yet) evolution. In JITIK there is one distribution level, as we have not yet considered ontology distribution between different JITIK "clusters". Nevertheless, FRODO does not have provisions for fine-tunning the ontology degree of distribution, which is one of the main qualities of JITIK ontology handling.

Discussion

Although they share the same basic ideas, the first proposed architecture and the prototype explore slightly different technological options, giving this way a range of possible solutions for specific systems.

The conceptual architecture illustrated in Figure 2 uses explicit persistent storage, as well as separation between schema and instances. This could be preferable over more homogeneous schemae like Jena in the case of extremely big instance numbers, because we can take advanage of efficient database queries, instead of specialized ontology inference mechanisms.

Our prototype does not use any form of persistent storage, though the Jena toolkit has recently offered persistance support. So, incorporating persistence is mainly a matter of updating our Jena version. But persistence is not essential for the COC at client agent side, as the client could load the base ontology as it is done in the prototype, and get additional definitions from the persistent storage on the OA side as we explained above. But of course, if the ontology is going the be enriched by the client agents, new concepts definitions should be stored permanently either in a local permanent storage at the COC, or sent to the OA in order to enrich the common ontology.

Conclusions

We have presented an architecture and a prototype which solve the ontology handling problem for the JITIK system, and which could be applied to other systems as well. The main requirements to apply our architecture is that there should be a common ontology, which is in principle agreed over the entire system, but which is not completely known by each agent in the system. So, we proposed a way of sharing the knowledge of the common ontology residing at an Ontology Agent, but avoiding the bottlenecks that would results from a centralized ontology handling. For this, we have incorporated to all the agents in the system a Client Ontology Component, which is capable of solving locally part of the ontology queries. This hybrid system could allow to better scale the system size.

We have used open standards for representing ontologies, like DAML+OIL. Further, we combined these standard formats with a multiagent-specific format offered by the JADE agent building toolkit.

A prototype is reported, which implements the basic elements of our architecture, making extensive use of the

Jena toolkit. A package (xont) was developed encapsulating all the additional functionality required to query the DAML+OIL ontologies from JADE.

Our hybrid approach, combining the advantages of a global and those of a local handling, introduces the possibility of fine-tuning the compromise between central an distributed ontology access, basically varying the size of the local ontologies. In one extreme, a zero size of COC ontology is equivalent to a central solution, whereas a COC ontology identical to the OA one gives a completely decentralized solution. Any intermediate solution is possible in principle.

The experiments carried out with our prototype demonstrate the basic querying and inferencing capabilities.

Future work

We intend to further explore the following aspects:

- We need to complete our quantitative experiment to show that our method outperforms both completely centralized and completely distributed approaches, considering time and space tradeoffs.
- In practice we could not allow unlimited updating of ontologies in the COC. One solution we foresee is to maintain a cache of the most frequently used definitions, eventually replacing the least used, or the least "valuable" ontology items.
- Define a mechanism for maintaining synchronization and coherence between central and distributed ontologies updating OA ontologies from additions to client ontologies.

References

Alexaki, S.; Athanis, N.; Vassilis, Ch.; Karvounarakis, G.; Maganaraki, A.; and Plexousakis, D. 2002. The ICS-FORTH RDFSuite: High level scalable Tools for the Semantic Web. Poster Session of the *Eleventh International World Wide Web Conference (WWW'02)*. Honolulu, Hawaii, USA.

Baader, F.; Calvanese, D.; McGuiness, D.; Nardi, D.; and Patel-Schneider, P. 2003. *The Description Logic Handbook*: Cambridge University Press.

Bechhofer, S.; Horrocks, I.; Goble, C.; and Stevens, R. 2001. OilEd: A Reason-able Ontology Editor for the Semantic Web, *24th German / 9th Austrian Conference on Artificial Intelligence.*

Bellifemine, F.; Caire, G.; Poggi, A.; and Rimassa G. 2003. *JADE - A White Paper* http://sharon.cselt.it/projects/jade/papers/WhitePaperJADEEXP.pdf

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. *The Semantic Web*, Scientific American, May.

Brena, R.; Aguirre, J.L.; and Trevino, A.C., Just-in-Time Knowledge Flow for Distributed Organizations using agents technology, *Knowledge Technologies 2001 Conference*, Austin, Texas, 4-7 March 2001, http://www2.gca.org/knowledgetechnologies/2001/proceedings/index.asp.

Caire, G. 2002. *JADE Tutorial. Application-Defined Content Languages and Ontologies.* http://sharon.cselt.it/projects/jade/doc/CLOntoSupport.pdf

Ceballos H. 2003. *Manejo de Ontologias en Sistemas Multiagentes por medio de un Agente de Ontologias aplicado a JITIK*. Master thesis Monterrey Tech.

Conolly, D.; Van Harmelen F.; Horrocks I.; McGuinness D.; Patel-Schneider P.; and Stein L. 2001. *DAML+OIL (March 2001) Reference Description.* W3C Note 18 Dec http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218

Collis, J.; and Ndumu, D. 1999. *The Zeus Agent Building Toolkit. The Role Modelling Guide.* British Telecommunications Report, http://more.btexact.com/projects/agents.htm

Dean M.; Schreiber G.; Van Harmelen F.; Hendler J.; Horrocks I.; McGuinness D.; Patel-Schneider P.; and Andrea Stein L. 2003. *OWL Web Ontology Language Reference.* W3C Working Draft, 31 March 2003. http://www.w3.org/TR/2003/WD-owl-ref-20030331/

Gandon, F. 2003. Agents handling annotation distribution in a corporate semantic web. *Web Intelligence and Agent Systems.* 1(1):23-45. OIS Press.

Lassila, Ora, and R. Swick, Ralph. 1999. *Resource Description Framework (RDF) Model and Syntax Definition Specification.* W3C Recommendation. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222

Lenat, D. 1993. Context dependence of representations in CYC. *In proceedings of Colloque ICO'93*. Montreal, Canada.

Liebowitz, J.; and Wilcox L.C. eds. 1997. *Knowledge Management and its Integretive Elements*. CRC Press.

McBride, B. *Jena Semantic Web Toolkit - Data Sheet.* http://www.hpl.hp.com/semweb/jena-datasheet.htm.

Maedche, A.; Motik B.; Stojanovic L.; Studer R.; and Volz R. 2003. An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies. *In proceedings of WWW2003*. Budapest, Hungary: ACM.

Nwana, H.; and Ndumu, D. 1999. A Perspective on Software Agents Research. *The Knowledge Engineering Review*. 14(2):1-18.

Ontoprise. http://www.ontoprise.com/

Van Elst, L.; and Abecker, A. 2002. Domain Ontology Agents in Distributed Organizational Memories. *Knowledge Management and Organizational Memories.* Rose Dieng-Kuntz and Nada Matta eds. Kluwer Academic Publishers.

Weichhardt, F.; Fillies, C.; and Smith, R. 2002. *The Semantic Web is the Database: Decentralised Modeling with central Coordination*. http://www.semtalk.com/pub/spain2.htm.

Wooldridge, M. 2002. *Introduction to MultiAgent Systems*. John Wiley and Sons.