Finding compromises between Local and Global Ontology Querying in Multiagent Systems

Hector Ceballos and Ramon Brena

Center for Intelligent Systems
Tecnologico de Monterrey, Mexico
{ceballos, rbrena}@itesm.mx

Abstract. As Ontologic knowledge gets more and more important in agent-based systems, its handling becomes crucial for successful applications. In the context of agent-based applications, we propose a hybrid approach, in which 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. In this sort of "caching" scheme, most frequent ontologic queries tend to remain stored locally. 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 as well a working prototype implementing our proposal, and discuss some performance figures.

1 Introduction

It is widely accepted that communication is an absolute requirement for most of the 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 [1].

The term *ontology* refers to a definition of meanings for terms used in inter-agent communications [2]. Ontologies allow to define concepts and their relations, properties, operations and the like in a structured fashion. Open standards like DAML- OIL[3], allow to publish ontologic knowledge in a way understandable both by humans and machines. But 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 [4] suggest to build huge centralized repositories of encyclopedic knowledge. Others considered this impractical in terms of performance and robustness, and prefer descentralized approaches [5]. But handling distributed ontologies generates new difficult problems as well, namely: 1) How to distribute the knowledge; 2) How to maintain some degree of coherence

among the different pieces of ontological knowledge. Further, independent partial ontology repositories could evolve independently and diverge, so in order to achieve a meaningful conversation we must put in place consensus mechanisms.

The method we will present in this paper for handling ontologic knowledge gives a "hybrid" local - global solution to this problem in the context of the JITIK project [6].

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 function. Although our ontology-handling proposal was primarily intended for its application in the JITIK system, our proposal is applicable in principle to a wide range of agent-based systems.

1.1 Our Approach

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. 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. They use their local ontologies, handled by the COC, as long 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 allows the agent to continue working. The COC stores locally the ontology addition so it could be used later.

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. Putting a piece of knowledge in either the OA or the COC has no catastrophic consequences, and it becomes just a matter of efficiency; adjustements are made as agents' operation proceed.

Of course, the solution we are presenting is valid only in some environments and not in others. In particular, the requirement for a global coherent ontology rules out open environments where there could be different or even contradictory definitions for similar items. But in restricted environments like, for instance, a given entreprise, this approach is feasible and efficient.

In section 2 we detail our method. Section 3 describes a working prototype. Experimental results are given in section 4; in section 5 we compare with other approaches; discussion and conclusions are given in sections 6 and 7.

2 The proposed solution for Ontology Handling in JITIK

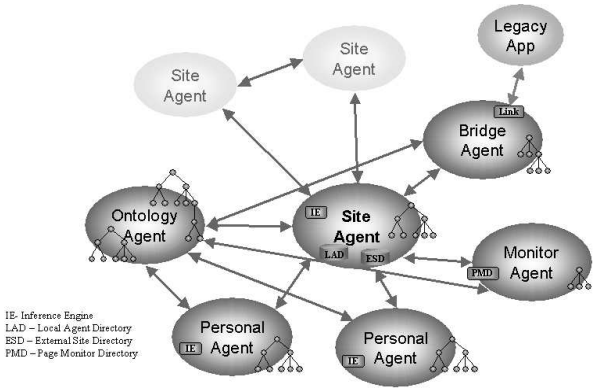

Fig. 1. JITIK agents

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. Actually, in this paper we will classify the agents in two categories: *ontology agents*, and "*regular*" agents, which are all the other agents, like personal agents, site agents, etc. Along this paper, regular agents are called *client agents* too because they are clients of the Ontology Agent.

2.1 Ontology Agent and Clients

Client agents try to fulfill their ontology 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.

Ontology Agent 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 for the client agents. Translation is sometimes necessary because the encoding for storing knowledge and answering queries, which is

mandated by performance requirements, is not the same as the one used in the client agents, which requires to be as light as possible. 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 Agent Client agents access ontology definitions through their COC. At startup they load a *base ontology*, and try to use it as long as it suffices for agent's work. In the JADE system [7], ontologies are needed for message validation purposes. 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.

2.2 Query Mechanism

In the following we present the query mechanism from client agents to the OA. It consists of three elements: the Query language, the Query Engine and the Answer Format.

One of the simplest query languages we have studied was RQL [8], that although is oriented to RDF, its syntax is similar to SQL, so query codification is not difficult.

The *Query Engine* is responsible for solving the queries made to the ontology. Its performance will be one of the most critical factor in the global performance of the OA, as it could be constantly answering questions coming from client agents. One such Query engine is RSSDB [9], which receives queries in RQL. We found that RSSDB's performance is reasonable.

Query responses are coded in a *response format*. Once the client agent receives an answer from the OA, it can process its information. This processing - decoding could be costly both for the client agent and for the OA if adequate formats are not chosen. Once more, we stress the need for limiting the transformations and interfaces used in these processes.

Among the response formats we found available, there are RDF over XML, and the *frames* format provided by the JADE ontology support. RSSDB gives answers in XML, so the translation to JADE frames should be done, either on the server or on the client side. We considered preferable to make the translation on the server side (the OA), because this way the process becomes transparent to client agents, and thus a replacement of technology on the OA does not need to be noticed in the client agents. As we wanted to use the RSSDB engine, translation was necessary, from JADE frames to RQL, for the client agent's queries, and from XML to frames to translate the OA's responses.

2.3 The COC

In our hybrid global-local approach, the client agents can access part of the ontology locally, or remotely, asking directly 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.

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.
- *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 containe*r *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.* 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.

3 Prototype

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

- Although we have worked with the RDFSuite package, and we actually built a very basic prototype with this technology, it did not supported full DAML+OIL ontologies. So we would need to develop the translations mentioned before from JADE schemas to RQL. We decided -at least provisionally- to use just the Jena package [10] instead, which at some point of our project incorporated persistent storage, thus becoming a very attractive solution. So RDFSuite was dumped altogether.
- 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 implements both the COC and the OA.

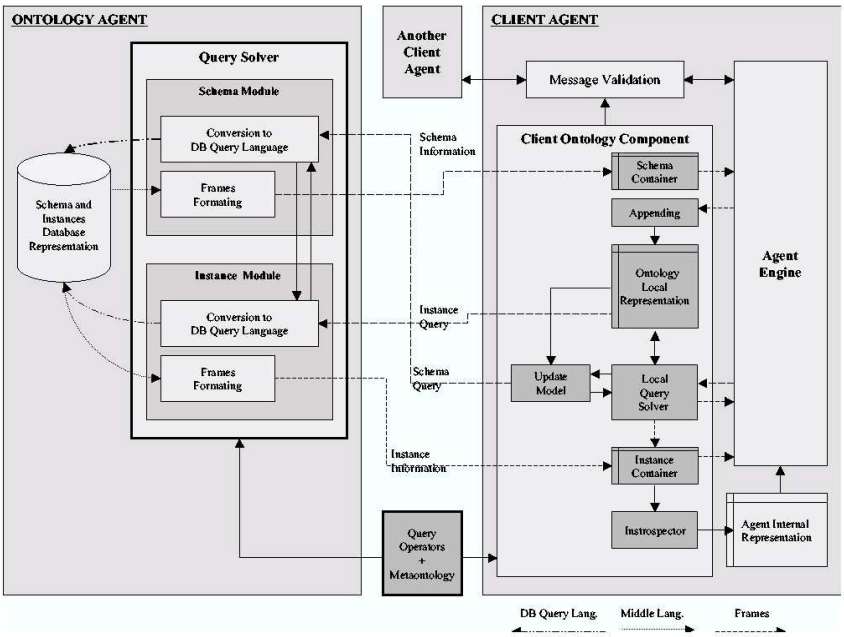

Fig. 2. Ontology handling architecture

- The COC does not automatically 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.

3.1 Query Solving

In the prototype, 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:

```
(ALL
  :VARIABLE (Variable :VALUETYPE CLASS_PROPS :NAME x)
  :PROPOSITION (DESCWHERE
    :DESC (CLASSDESCRIPTOR :CLASS_PROPS
      (Variable :VALUETYPE CLASS_PROPS :NAME x))
    :WHERE (CLASSDESCRIPTOR :CLASS_NAME Worker)))
```

Using our example ontology, the obtained query result is as follows:

```
(RESULTS :RESULTS_SET (DESCLIST
  #0 (CLASSDESCRIPTOR
    :CLASS_NAME Worker
    :CLASS_PROPS (PROPLIST
      #0 (PROPERTYDESCRIPTOR :PROP_NAME id)
      #1 (PROPERTYDESCRIPTOR :PROP_NAME responsability)
      #2 (PROPERTYDESCRIPTOR :PROP_NAME email)
      #3 (PROPERTYDESCRIPTOR :PROP_NAME name)))))
```

We can see that the class Worker groups in the result the list of the properties we asked about. We included the class name so that this answer is self-contained,

and could be incorporated to the local ontology at the client agents in a meaningfull way. It shoud be noted that the returned property list in this example includes not only the direct properties of Worker, but those defined in its superclasses as well.

It is left to the programmer to interpret and use the results given by the ontology facilities in our prototype. The system only carries out automatically the merging of the arriving responses with the local ontology, as is discussed in the next subsection.

3.2 Adapting JADE Ontology Support

From version 2.5, JADE incorporates some support for ontology handling [11]. 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, which are called *frames*.

3.3 Local Ontology Extensibility

In the prototype we achieve basic 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.

As we shall discuss later, the COC extensibility would have to be bounded in some way, as an arbitrary growth would either overflow the COC or at least make it similar in size to the OA.

4 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. We assumed 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 [12].

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 are accessible by internet. We used the tool OilEd [13] to edit our test ontology.

The main result from this experiments was that 100% of a sample of 15 queries were correctly answered. A greater number of tests was considered unneeded, because of the 100% success, and because a high level of redundancy was evident as more complex queries were formulated. These experiments basically ensured that the prototype is correctly built.

Additionally, we carry out a simulation to evaluate our approach performance. We used a set of 200 fixed queries over an ontology of 4000 elements and supposed having a queries solving algorithm with $log\ n$ complexity. We simulate a scenario where a client agent randomly generates queries and the probability that the query can be locally solved depends on the local ontology size.

Over this scenario we observed a strong dependency on the ratio between the local solving time and the remote solving time. The slower the remote response, bigger the gaining in performance, and this is accentuated once 50% or more of the ontology is transferred to the COC. Figure 3 shows normalized times on three experiments rounds where the rate between the average remote time and the average local time is denoted by r (smaller r means slower remote response).

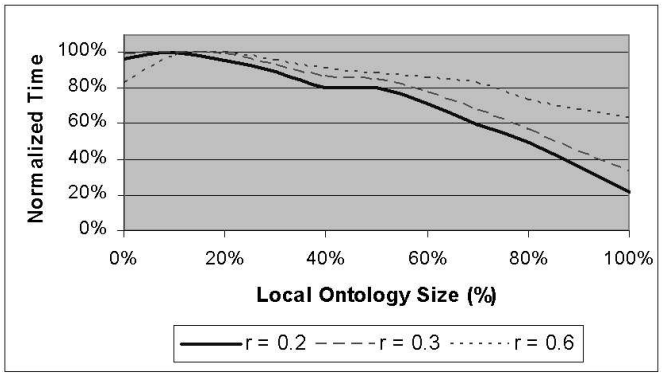

Fig. 3. Time Performance.

We measure efficiency with the product of the total time required to evaluate a queries serie and the space used to store the local ontology; smaller values means better performance. Both variables were normalized giving them the same importance. This measure only considers agent individual performance.

Meanwhile local and remote response times keeps similar ($r > 0.5$), efficiency factor grows constantly until reach the complete transference of the ontology to the client agent, wich means we obtained no gaining on this transference. Nevertheless, when the remote response time becomes slower than local we observed a

curve in the efficiency with high values in the middle of spectrum. This denotes that individual performance is good meanwhile just a small portion of the ontology is locally allocated, after this point efficiency decrease. At the end, when ontology is fully locally stored, efficiency factor improves until a fixed value, gave by the local time response.

In Figure 4 this behavior can be observed. We would use the efficiency with full local ontoloy to limit the growth of the local ontology before efficiency degrades. This way, at figure 4 we could observe that limit for r=0.2 would be 23% of ontology locally allocated.

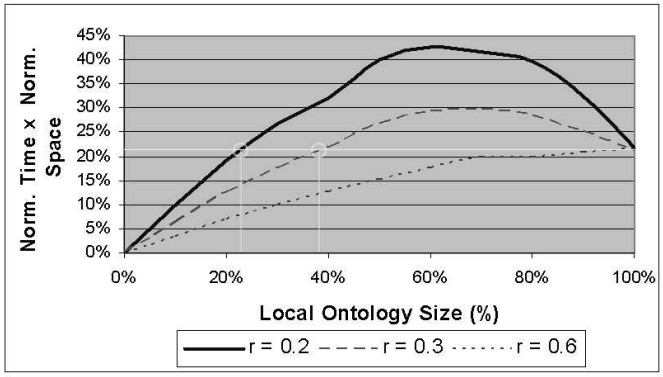

Fig. 4. Efficiency (time x space).

We also simulated the use of a queries cache. In experiments we worked with a 10 queries cache and vary the number of available queries. Query selection is given by a normal distribution. The average cache utilization was 16%, 8% and 3% for a cache equals to 10%, 5% and 2% of the number of available queries, respectively. The behavior of the improvement gained with the use of this cache as the ontology is transferred to the COC can be observed in Figure 5, where the cache size is 5%.Even when the response time becomes zero for queries stored in cache, space required for storing this queries impacts negatively into the efficiency factor, so we must find a threshold in this aspect too.

5 Related Work

In the KAON project [14] they stress reusing existing RDF ontologies and propagate the changes to distributed ontologies. The ontologies URIs are tracked in an *ontology registry* kept by the *ontology servers*, which take care of the ontology evolution (inclusion, updating, changes propagation, etc.). Each Ontology Server provides query resolution service to its agent community and the ontology evolution is driven by inter-ontology servers communication. The original ontology

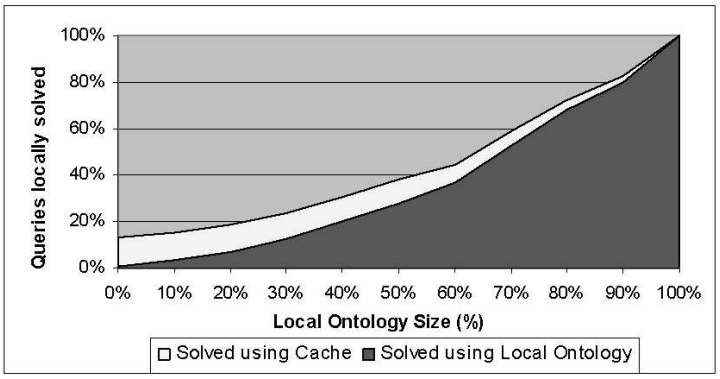

Fig. 5. Improvement using a Queries Cache.

URI is kept unchanged, and a local copy of the ontology is used in the Ontology Server for query resolution.

Our ontology agent, in contrast, stores full DAML+OIL ontologies. still has to improve in the ontology evolution aspect. On the minus side, we have not taken into account yet ontology evolution, mainly because this would introduce very hard consistency-checking problems. In our approach, updating is made at the client agents, through the COC.

In COMMA [15], as in JITIK, a global ontology is propagated over the entire agent society. Each agent has a complete copy of the ontology/model and can solve queries by itself. COMMA uses RDF for representing ontologies. They designed an API with downloads, updates and querying mechanisms for other agents. This approach is good for small ontologies that do not change quit often. In the society, the *Ontology* Archivist is responsible for maintaining and accessing the ontology. Obviously this approach lies at the centralized side of the spectrum.

FRODO [16] considers *ontology provider* and *ontology consumer* roles. Providers concentrate provision of ontology services as well as the acquisition and maintenance of the domain ontology. Consumers use domain ontologies in order to execute a specific application. Providers operate in FRODO at two levels: internally to a subsystem (with *Domain Ontology* Agents), and externally or inter-systems (with *Distributed Domain Ontology* Agents). In JITIK we have only developed the internal level with the OA and the COC, though conceptually we consider interagent communication through the "site agents" connection. FRODO defines three categories of competencies: Ontology *Use*, Ontology *Evolution* and Ontology *Socialization*. In JITIK we considered just the first and the third.

6 Discussion

Although they share the same basic ideas, the 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 expliclit persistent storage, as well as separation between schema and instances. This could be preferable over more homogeneous schemes 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 we think persistence is not essential for the COC at client agent side; 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.

7 Conclusions

We have presented an architecture which solve the ontology handling problem for the JITIK system, and which could be applied to other systems as well.

The main requirement 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.

We have used standard 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.

We think our hybrid approach 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 COC ontology is equivalent to a central solution, whereas a COC identical to the OA gives a completely decentralized solution. Any intermediate solution is possible.

The experiments carried out with our prototype demonstrate the basic querying and inferencing capabilities. Simulation presented is still preliminar and will be focused on measuring performance of the centralized-distributed approaches.

7.1 Future Work

One validation that we still have to carry out is to show that, in terms of global efficiency, our approach outperforms both only-global as well as only-local ontologies, assumming a global coherent ontology, as we mentioned previously.

It is also important to test our method in a wide range of real knowledge-intensive multiagent scenarios, in such a way that the global-local fine tunning we mentioned before could be put in practice.

Another aspect is that continuous incorporation of knowledge pieces to the COC coming from the OA would eventually overflow the COC. One solution we foresee is to maintain a "cache" of the most frequently used definitions, eventually replacing the least used.

Even when common ontology is selected arbitrarily at beginning, another ontology can be selected through the system evolution based on use statistics. Analysis on requests to the Ontology Agent could help to identify the most frequently requested elements, meanwhile statistics on the local ontology can help to mark the less important elements. Going beyond, common ontology can be modeled by the Ontology Agent based on these statistics and constructs variants of the common ontology for agents groups identified by clustering techniques.

References

1. H. S. Nwana and D. T. Ndumu, "A perspective on software agents research," *The Knowledge Engineering Review*, vol. 14, no. 2, pp. 1–18, 1999.
2. M. J. Wooldridge, *Multi-agent systems : an introduction*, Wiley, Chichester, 2001.
3. I. Horrocks, "DAML+OIL: a description logic for the semantic web," *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 25, no. 1, pp. 4–9, Mar. 2002.
4. D. B. Lenat, "Computers versus common sense," in *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 29-31, 1991*, James Clifford and Roger King, Eds. 1991, p. 1, ACM Press.
5. C. Fillies, G. Wood-Albrecht, and F. Weichhardt, "Pragmatic applications of the Semantic Web using SemTalk," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 42, no. 5, pp. 599–615, Aug. 2003.
6. R. Brena, J. L. Aguirre, and A. C. Trevino, "Just-in-time knowledge flow for distributed organizations using agents technology," in *Proceedings of the 2001 Knowledge Technologies 2001 Conference, Austin, Texas, 4-7 March 2001*, 2001.
7. M. C. Rinard and M. S. Lam, "The design, implementation, and evaluation of Jade," *ACM Transactions on Programming Languages and Systems*, vol. 20, no. 3, pp. 483–545, 1 May 1998.
8. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl, "Rql: A declarative query language for rdf," In The 11th Intl. World Wide Web Conference (WWW2002), http://citeseer.nj.nec.com/556066.html, 2002.
9. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle, "The rdfsuite: Managing voluminous rdf description bases," Technical report, Institute of Computer Science, FORTH, Heraklion, Greece, http://www.ics.forth.gr/proj/isst/RDF/RSSDB/rdfsuite.pdf.", 2000.

10. HP Labs, "Jena semantic web toolkit - data sheet," http://www.hpl.hp.com/semweb/jena-datasheet.htm, 2000.
11. G. Caire, "Jade tutorial: Application-defined content languages and ontologies," http://sharon.cselt.it/projects/jade/doc/CLOntoSupport.pdf, 2002.
12. H. Ceballos, "Disign and implementation of an ontoloty agent in the jitik project," M.S. thesis, Tecnologico de Monterrey, Monterrey, Mexico, June 2003.
13. I. Horrocks, "DAML+OIL: a reason-able web ontology language," in *Proc. of EDBT 2002*. Mar. 2002, number 2287 in Lecture Notes in Computer Science, pp. 2–13, Springer, 2002.
14. D. Oberle, R. Volz, B. Motik, and S. Staab, "An extensible ontology software environment," in *Handbook on Ontologies*, International Handbooks on Information Systems, chapter III, pp. 311–333. Steffen Staab and Rudi Studer, Eds., Springer, 2004.
15. C. Consortium, P. Perez, H. Karp, R. Dieng, O. Corby, A. Giboin, F. Gandon, J. Quinqueton, A. Poggi, and G. Rimassi, "Corporate memory management through agents," http://citeseer.ist.psu.edu/consortium00corporate.html", 2000.
16. L. Van Elst and A. Abecker, "Domain ontology agents in distributed organizational memories," http://citeseer.ist.psu.edu/vanelst01domain.html, 2001.