A Framework for Distributed Ontology Systems

Xueying Chen[1] and Michel Dumontier[1,2]

[1] School of Computer Science,
[2] Department of Biology,
Carleton University, 1125 Colonel By Drive, K1S 5B6, Ottawa, Canada
xychen@scs.carleton.ca, michel_dumontier@carleton.ca

Abstract. In recent years, both the number and size of web ontologies has been fervently growing. Although much attention has been paid to ontology integration so as to enable interoperability between different ontologies, the problem of how to handle ontologies of large size has not been sufficiently addressed. In this paper, we propose a general framework for peer-to-peer distributed ontology systems which manages large ontologies size by partitioning an ontology into ontology fragments and distributing them over multiple autonomous nodes. Taking advantage of peer-to-peer computer networks, a distributed ontology system promises distributed reasoning as well as collaborative ontology development.

1 Introduction

In recent years, we have witnessed both the number of ontologies appearing on the web growing rapidly and the size of ontologies expanding dramatically. An increase in the number of ontologies poses problems in the querying of multiple ontologies that are possibly heterogeneous and independently developed. Ontology integration systems(OIS) [4] [16] provide solutions to these problems by semantically combining the information in different ontologies, and providing the user with a unified view through which the user can query those combined ontologies. In contrast, increases in the ontology size often compromises the efficiency of query processing and results in slow response times. For example, consider ontology classification which requires the generation of the complete class hierarchy from subclass relations between every named class, for a large ontology such as SNOMED[3], the CEL[4] reasoner requires nearly half an hour, FaCT++[5] requires more than an hour, and RacerMaster[6] and Pellet[7] fail to complete the task due to memory exhaustion [2][8]. These limitations could be overcome if the query processing were distributed across a set of nodes.

[3] http://www.snomed.org, an important bio-medical ontology that contains estimated 300,000 concepts in January 31, 2004 [19].
[4] http://lat.inf.tu-dresden.de/systems/cel
[5] http://owl.cs.manchester.ac.uk/fact++
[6] http://www.racer-systems.com/products/index.phtml
[7] http://clarkparsia.com/pellet
[8] In [2], experiments were performed on a PC with 2.8GHz Intel Pentium 4 processor and 512MB memory running Linux v2.6.14.

The design and maintenance of increasingly large ontologies often requires a team of ontology engineers to work collaboratively [17]. In collaborative environments, changes made by any one individual must be propagated while minimizing conflicts in terms of simultaneous editing as well as in the introduction of semantic inconsistencies. Most collaborative ontology editing systems use a client/server model, thereby centralizing the problem of synchronization and consistency checking. However, this approach does not address the issue of query processing efficiency.

In this paper we propose a novel Distributed Ontology System (DOS) that aims to solve query processing issues resulting from the large ontology size while addressing the needs in distributed ontology design. The key idea is that the large ontology is partitioned into smaller pieces, called ontology fragments, and these fragments are distributed across an arbitrary number of autonomous nodes. Although a query may be posed to any node, the master node is the one which accepts the query and subsequently distributes the computation for evaluating the query to other nodes who process those computational tasks locally and return the results to the master node. The master node then integrates the results and returns the final answer to the user. We assume that the ontology is based on description logics and is written in OWL, due to its widespread adoption as an ontology language for the semantic web.

DOS also provides consistency and synchronization for collaborative ontology edit. The ontology may be edited on any node and change propagation is transparent to the user. Changes to ontology fragments that require re-distribution are propagated to a selected set of other nodes by the master node. Thus, unlike existing client/server approaches, the DOS model does not require a centralized ontology, and is expected to be more scalable and responsive.

The paper is organized as follows: In section 2, we set up a formal framework for distributed ontology systems. In section 3 and 4, we describe approaches for partitioning ontology, and discuss problems such as query processing and collaborative ontology development. Section 5 compares related works, while conclusions and future work are in section 6.

2 Framework for a Distributed Ontology System(DOS)

In this section, we set up a formal framework for distributed ontology systems. Given a large ontology, we divide it into ontology fragments and distribute the fragments over a number of autonomous nodes. These fragments are components of the same ontology and therefore logically interrelated. Their relationship should be recorded so that it can be used to coordinate the nodes to complete tasks such as query processing, collaborative ontology development and so on. A major challenge in the design of a distributed ontology system is the partitioning of the ontology, which has significant impact on the efficiency of the system, especially the efficiency of query processing. Consequently, the metadata(semantic information) about each fragment is important and should be taken into account in formalizing a distributed ontology system.

We summarize three main components of a distributed ontology system: a large ontology, ontology fragments into which the large ontology is partitioned, and a dictionary contains metadata about each fragment. In the rest of this paper, we will use the traditional description logic notation [1] [13] for the sake of compactness.

Formally,

Definition 1. *A distributed ontology system $\mathcal{I}$ is a triple $< \mathcal{O}, \mathcal{F}, \mathcal{D} >$ where*

1. *$\mathcal{O}$ is an ontology, expressed in a language $\mathcal{L}_{\mathcal{O}}$ over an alphabet $\mathcal{A}_{\mathcal{O}}$ containing terms of $\mathcal{O}$.*
2. *$\mathcal{F}$ is a set of ontology fragments $F_1, \cdots, F_n$ into which $\mathcal{O}$ is divided. We denote with $\mathcal{A}_{F_i}$ the alphabet containing terms of ontology fragment F_i. Here, $\mathcal{A}_{F_i} \subseteq \mathcal{A}_{\mathcal{O}}$, and $\mathcal{A}_{\mathcal{O}} = \bigcup_{i=1,\cdots,n} \mathcal{A}_{F_i}$.*
3. *$\mathcal{D}$ is a dictionary which contains the semantic information of each fragment.*

We can view the ontology $\mathcal{O}$ and each fragment F_i as theories from a logical point of view. Note that ontology fragments are not required to be mutually disjoint.

As for the semantics of a distributed ontology system $\mathcal{I}$, we illustrate it from two directions. First, we start from a model $M_{\mathcal{O}}$ of the ontology $\mathcal{O}$, i.e., an interpretation that satisfies the theory $\mathcal{O}$, from the definition, we know that theories $F_i (i = 1, \cdots, n)$ are subsets of $\mathcal{O}$, thus, $M_{\mathcal{O}}$ is also a model for all ontology fragments F_i. On the other hand, given a model M of an ontology fragment F, there are only two cases that make M a model of $\mathcal{O}$: (1) $\mathcal{O}$ and F coincide, (2) M is a model for all ontology fragments $F_i (i = 1, \cdots, n)$.

We now specify the semantics of queries posed to a distributed ontology system $\mathcal{I}$. In $\mathcal{I}$, queries are posed to ontology $\mathcal{O}$ instead of ontology fragments F_i, therefore, they are expressed in a query language $\mathcal{Q}_{\mathcal{O}}$ over the alphabet $\mathcal{A}_{\mathcal{O}}$. In general, if q is a query of arity n and M is model of $\mathcal{O}$, we denote with q^M the set of tuples (of arity n) in M that satisfy q.

We assume ontologies are expressed in OWL-DL, which is founded in description logics. Correspondingly, we classify the knowledge in an ontology into two classes: axioms(TBox and RBox which contain terminological axioms and role axioms, respectively) and facts(ABox which contains instances). If the size of the ontology is largely attributable to its collection of facts, then the partitioning of facts across a network should have benefits in terms of reasoning. We identify two categories of ontology fragments:

1. Ontology fragments containing partial axioms and partial facts, that is, each fragment contains an incomplete TBox, RBox and ABox.
2. Ontology fragments containing complete set of axioms and partial facts, that is, each fragment contains an complete TBox and RBox, but incomplete ABox.

We can distinguish two types of ontology fragments from a semantics perspective:

1. Modular ontology fragments with respect to a certain alphabet $\mathcal{A}_k$, that is, given a query q constructed using terms in $\mathcal{A}_k$, instead of evaluating q against the ontology $\mathcal{O}$, the user obtains the same answers by evaluating q against a modular fragment F.
2. Non-modular ontology fragments, namely, fragments that are not modular with respect to any terms.

In following sections, we will discuss problems of query processing and ontology development in each case of partitioning.

3 Partial Axioms and Partial Facts

In this section, we discuss query processing and ontology development when each ontology fragment, whether modular or non-modular, contains an imcomplete TBox, RBox and ABox.

3.1 Modular Ontology Fragments

We illustrate modular partial ontology fragments through an example.

Example 1. Consider an ontology $\mathcal{O}$ as showed in Table 1, where in TBox, the concept $FemaleStudent(MaleStudent)$ is defined as the conjunction of concepts $Female(Male)$ and $Student$, concepts $FemaleStudent$ and $MaleStudent$ are disjoint. The axiom in RBox says that the role $Attends$ can be applied to instances of concept $Course$ only. ABox contains the set of facts.

Table 1. Ontology $\mathcal{O}$

TBox	$FemaleStudent \equiv Female \sqcap Student$	
	$MaleStudent \equiv Male \sqcap Student$	
	$FemaleStudent \sqcap MaleStudent \sqsubseteq \bot$	
RBox	$\exists Attends.\top \sqsubseteq Course$	
ABox	$Student(Ann)$	$Female(Ann)$
	$Student(Bob)$	$Male(Bob)$
	$Student(Carol)$	$Student(Daisy)$
	$Course(SCS1001)$	$Course(SCS3005)$
	$Attends(Ann, SCS1001)$	$Attends(Carol, SCS1001)$
	$Attends(Bob, SCS3005)$	$Attends(Daisy, SCS3005)$

We divide $\mathcal{O}$ into two fragments F_1 and F_2 as showed in Table 2 and 3, respectively.

Given queries

$$q_1(x) := Student(x)$$

and

$$q_2(x) := Course(x)$$

Evaluating $q_1(x)$ against F_1 returns the same answers as evaluating it against $\mathcal{O}$, i.e., $\{Ann, Bob, Carol, Daisy\}$. We say fragment F_1 is modular with respect to the alphabet $\mathcal{A}_{\mathrm{k}} = \{Student\}$.

Evaluating $q_2(x)$ against F_2 also returns the same answers as evaluating it against $\mathcal{O}$, i.e., $\{SCS1001, SCS3005\}$. We say fragment F_2 is modular with respect to the alphabet $\mathcal{A}_{\mathrm{k}'} = \{Course\}$.

Table 2. Ontology Fragment F_1

TBox	$FemaleStudent \equiv Female \sqcap Student$	
	$MaleStudent \equiv Male \sqcap Student$	
	$FemaleStudent \sqcap MaleStudent \sqsubseteq \bot$	
ABox	$Student(Ann)$	$Female(Ann)$
	$Student(Bob)$	$Male(Bob)$
	$Student(Carol)$	$Student(Daisy)$

Table 3. Ontology Fragment F_2

RBox	$\exists Attends.\top \sqsubseteq Course$	
ABox	$Course(SCS1001)$	$Course(SCS3005)$
	$Attends(Ann, SCS1001)$	$Attends(Carol, SCS1001)$
	$Attends(Bob, SCS3005)$	$Attends(Daisy, SCS3005)$

"*Modular*" is from term "*module*" which is first defined for the purpose of modular reuse of ontologies [10] [11]. Intuitively, a fragment F of an ontology $\mathcal{O}$ is modular with respect to an alphabet $\mathcal{A}_{\mathrm{k}}$ if it captures the knowledge about terms in $\mathcal{A}_{\mathrm{k}}$ such that evaluating queries about terms in $\mathcal{A}_{\mathrm{k}}$ against $\mathcal{O}'$ returns same answers as evaluating the queries against $\mathcal{O}$.

Formally,

Definition 2. *Let $\mathcal{O}$ be an ontology and $\mathcal{A}_{\mathcal{O}}$ its alphabet, F a fragment of $\mathcal{O}$, and q a query posed over $\mathcal{O}$. We say F is modular with respect to an alphabet $\mathcal{A}_{\mathrm{k}}$ if it holds that*

$$\mathcal{O} \models q^{\mathcal{O}} \Longleftrightarrow F \models q^{\mathcal{O}}$$

for $\mathcal{A}_{\mathrm{q}} \subseteq \mathcal{A}_{\mathrm{k}}$ where $\mathcal{A}_{\mathrm{q}}$ is the alphabet of q.

Thus, at the stage of partitioning, an algorithm is required to check whether a fragment F_{i} is modular or not. If the result is positive, identify the relative alphabet $\mathcal{A}_{\mathrm{i}}$. The dictionary $\mathcal{D}$ records the correspondence between F_{i} and $\mathcal{A}_{\mathrm{i}}$($\mathcal{D}$ contains ontology metadata, which can be described using Ontology Metadata Vocabulary [14]). When a query q is posed over the ontology $\mathcal{O}$, the system looks up $\mathcal{D}$, compares $\mathcal{A}_{\mathrm{q}}$ with $\mathcal{A}_{\mathrm{i}}$. If $\mathcal{A}_{\mathrm{q}} \subseteq \mathcal{A}_{\mathrm{i}}$ found, q is forwarded to the node where the fragment F_{i} locates, and then evaluated against F_{i}.

While the modular ontology fragments are very efficient in query processing if a matched alphabet can be found, it is easy to see the disadvantages. One is that a matched alphabet for a given query is not guaranteed. If the system fails to find a modular fragment for a query, then the processing of such a query is a different (section 3.2). Another disadvantage is that the modular structure is vulnerable to edits of ontology. DOS provides collaborative ontology development by allowing the user to edit the ontology at any nodes, however, any changes to the ontology may result in destroying the modular structure, especially where the edited fragment and the modular fragment are not the same. For instance, a user can edit the ontology $\mathcal{O}$ in example 1 by adding a fact $\{Course(SCS2003)\}$ to the ABox of F_1, as a consequence, F_2 is no longer modular with respect to the term $Course$. In this case, the strategy of synchronization between fragments is important. When a change has been made to a fragment, the system determines to which nodes the change should be propagated so that the modular structure of fragments would not be compromised.

3.2 Non-Modular Ontology Fragments

In the case that each ontology fragment is not modular, and contains only partial axioms and facts, the dictionary $\mathcal{D}$ is of little help. Queries, in some case may need to be pre-processed, are forwarded to other nodes, evaluated locally, the answers are combined afterwards. The key here is that integration of the axioms is necessary in order to ensure the soundness of distributed reasoning.

Example 2. Given the ontology $\mathcal{O}$ in Example 1, we partition $\mathcal{O}$ into two fragments as showed in Table 4 and Table 5, respectively, evaluate the query

$$q_3(x) := FemaleStudent(x)$$

posed over $\mathcal{O}$.

Table 4. Ontology Fragment F_1

TBox	$FemaleStudent \sqcap MaleStudent \sqsubseteq \bot$	
RBox	$\exists Attends.\top \sqsubseteq Course$	
ABox	$Student(Ann)$	$Male(Bob)$
	$Student(Carol)$	
	$Course(SCS1001)$	$Course(SCS3005)$
	$Attends(Ann, SCS1001)$	$Attends(Carol, SCS1001)$

Evaluate q_3 against either F_1 or F_2 will return no result. We need to pre-process q_3 as follows.

First, we integrate the axiom part of fragments such that each fragment has the complete set of axioms of the ontology, as showed in Table 6 and Table 7. Then we unfold the concept in q_3 using the axioms:

$$FemaleStudent(x) \equiv FemaleStudent(x) \vee (Female(x) \wedge Student(x))$$

Table 5. Ontology Fragment F_2

TBox	$FemaleStudent \equiv Female \sqcap Student$	
	$MaleStudent \equiv Male \sqcap Student$	
RBox	$\exists Attends.\top \sqsubseteq Course$	
ABox	$Student(Bob)$	$Female(Ann)$
	$Student(Daisy)$	
	$Attends(Bob, SCS3005)$	$Attends(Daisy, SCS3005)$

We decompose q_3 into two subqueries and evaluate them against F_1' and F_2'.

$$q_3'(x) := FemaleStudent(x)$$
$$q_3''(y) := Female(y) \vee Student(y)$$

Finally, the master node collects the answers to subqueries and computes the final answer, which is the union of answers to q_3'(empty in this example) and the intersection of answers to q_3'', i.e., $\{Ann\}$.

Table 6. Ontology Fragment F_1'

TBox	$FemaleStudent \equiv Female \sqcap Student$	
	$MaleStudent \equiv Male \sqcap Student$	
	$FemaleStudent \sqcap MaleStudent \sqsubseteq \bot$	
RBox	$\exists Attends.\top \sqsubseteq Course$	
ABox	$Student(Ann)$	$Male(Bob)$
	$Student(Carol)$	
	$Course(SCS1001)$	$Course(SCS3005)$
	$Attends(Ann, SCS1001)$	$Attends(Carol, SCS1001)$

Table 7. Ontology Fragment F_2'

TBox	$FemaleStudent \equiv Female \sqcap Student$	
	$MaleStudent \equiv Male \sqcap Student$	
	$FemaleStudent \sqcap MaleStudent \sqsubseteq \bot$	
RBox	$\exists Attends.\top \sqsubseteq Course$	
ABox	$Student(Bob)$	$Female(Ann)$
	$Student(Daisy)$	
	$Attends(Bob, SCS3005)$	$Attends(Daisy, SCS3005)$

This partition is suitable for ontologies that have large ABoxes and simple TBoxes and RBoxes. In this case, collaborative ontology development becomes trivial. The user can edit ontology through any node without worrying about synchronization between fragments.

4 Complete Axioms and Partial Facts

In this section, we consider the case where each fragment contains an complete TBox and RBox, but partial ABox. We still distinguish two kinds of fragments: modular and non-modular.

4.1 Modular Ontology Fragments

This setting is similar to modular partial fragments in section 3.1. The difference is that each fragment contains complete set of axioms. Therefore, in case that a matched modular fragment for a given query is not found, the physical integration of axiom part is not attempted because each fragment already contains a complete set of axioms.

Example 3. Given the ontology $\mathcal{O}$ in Example 1, we now partition $\mathcal{O}$ into two fragments as showed in Table 8 and Table 9, respectively.

Table 8. Ontology Fragment F_1

TBox	$FemaleStudent \equiv Female \sqcap Student$	
	$MaleStudent \equiv Male \sqcap Student$	
	$FemaleStudent \sqcap MaleStudent \sqsubseteq \bot$	
RBox	$\exists Attends.\top \sqsubseteq Course$	
ABox	$Course(SCS1001)$	$Course(SCS3005)$
	$Attends(Ann, SCS1001)$	$Attends(Carol, SCS1001)$

Table 9. Ontology Fragment F_2

TBox	$FemaleStudent \equiv Female \sqcap Student$	
	$MaleStudent \equiv Male \sqcap Student$	
	$FemaleStudent \sqcap MaleStudent \sqsubseteq \bot$	
RBox	$\exists Attends.\top \sqsubseteq Course$	
ABox	$Student(Ann)$	$Female(Ann)$
	$Student(Bob)$	$Male(Bob)$
	$Student(Carol)$	$Student(Daisy)$
	$Attends(Bob, SCS3005)$	$Attends(Daisy, SCS3005)$

Fragment F_1 is modular with respect to $\{Course\}$, and F_2 is modular with respect to $\{Student\}$. However, consider query

$$q_4(x, y) := Attends(x, y)$$

where neither F_1 nor F_2 is modular with respect to it.

In this case, the system passes q_4 to both fragments. Query q_4 is evaluated against each fragment, corresponding answers are returned to the master node and then integrated to generate the final answers, i.e.,

$$\{(Ann, SCS1001), (Carol, SCS1001), (Bob, SCS3005), (Daisy, SCS3005)\}$$

As for the collaborative ontology development, we say any axiom change should be propagated to all the other fragments in order to maintain the completeness of TBox and RBox. Synchronization of a change to the ABox is required only when this change compromises the modular structure of fragments.

4.2 Non-modular Ontology Fragments

This case is similar to the case discussed in section 3.2, the only difference is now each fragment has complete set of axioms. Therefore, query processing is the same as that in section 3.2 after the integration of axioms. And synchronization is required only for the changes made to the axioms.

5 Related Work

There are three threads of research work that are related to our work: Peer-to-peer ontology integration systems(OISs), systems that provide collaborative ontology development service, and modular reuse of ontology.

OISs [4] [16] provide a unified view(global ontology) over a number of ontologies(local ontologies). An OIS should be able to (1) understant the similarities between two ontologies, i.e., mappings between ontologies, (2) represent these mappings in an appropriate formalism, and (3) use the mappings when performing reasoning tasks. They can be classified into three categories according to their mappings: global-as-view, local-as-view and peer-to-peer. Global-as-view mappings associate to each term in the global ontology a view expressed using terms in local ontologies [3]. In contrast, local-as-view, associate each term in a local ontology using terms in the global ontology [5]. In peer-to-peer OISs, mappings are defined between local ontologies [6] [7] [8] [12] [15].

DOS and peer-to-peer OIS are similar in the sense that they both provide distributed reasoning and have no central control, i.e., all nodes throughout the network act autonomously. In terms of distributed reasoning, taking full advantage of the underlying reasoners of each fragment, DOS is able to support the same set of queries and ontologies of same expressivity as those underlying reasoners. Unlike DOSs, OISs usually can handle only a limited set of queries and ontologies of a certain expressivity. For example, Drago [7] [8] is based on the distributed description logic, which extends standard description logic with bridge rules, a means for expressing semantic mappings between local ontologies. In Drago, reasoning with multiple ontologies is accomplished by a distributed tableau algorithm which combines local reasoning [20]. For SHIQ ontologies, Drago supports reasoning for terminological knowledge(Tbox and RBox), but

can only handle instance retrieval for assertional knowledge(ABox). Another peer-to-peer OIS, KAONp2p [12] [15], which uses views(conjunctive queries) to define mappings between local ontologies, can process only conjunctive queries for OWL-DL ontologies extended with DL-safe rules by converting them into positive disjunctive datalogs, and then forwarding the datalogs to all the nodes where the datalog is executed locally.

There are few works on collaborative ontology development, though none of them provides distributed reasoning at the same time. [18] describes ten different ontology engineering projects' infrastructure, architecture and workflows. These projects use a Client/Server mode where a centralized ontology is created and maintained on the server, and users edit the ontology via client ends. They provide two types of editing: asynchronous and synchronous. In asynchronous editing, multiple users download a copy of the ontology from the server, work on their copies, and then submit their changes to a central server, such as OpenGalen[9] and SNOMED. This type of editing requires that changes made by different users be integrated before being submitted, but inconsistency can be removed in this process. While in synchronous editing, users connect to the centralized ontology on the server and submit their changes directly to the ontology, such as ONKI[17], APSF-DDTF[10] and CLEF[11]. Synchronous editing allows the changes immediately taking effect and visible to other users, but inconsistency in the changes is hard to detect. Collaborative ontology development in DOS is similar to synchronous editing. The difference is that users interact with ontology fragments instead of a centralized ontology and leave the decision of synchronization to the system.

The research on modular reuse of ontology [9] [10] [11] focuses on extracting meaningful fragments from an ontology. For this purpose, the term "module" is defined as the minimal fragment that completely captures the meaning of a given set of terms, i.e., to include all axioms relevant to the meaning of these term. The problem of determining a module in an ontology is computationally expensive: *EXPTIME-complete* if the ontology is written in $\mathcal{EL}$, *2-EXPTIME-complete* in $\mathcal{ALCIQ}$, and *undecidable* in $\mathcal{ALCIQO}$ [11]. In DOS, we extend "module" to "modular" in order to isolate a small, but not necessarily minimal, fragment which captures the answers to a given set of queries. By releasing the condition of minimality, it should be possible to find algorithms of lower complexity for determining "modular" fragments in an ontology. However, there exist disadvantages of using modular fragments in DOS. One is inflexibility in the sense that a modular fragment is always related to a specific set of terms and can only answer queries about those term. Another disadvantage, similar to that described in section 3.1, is that the modular structure is vulnerable to edits.

[9] http://www.opengalen.org
[10] http:// www.apsf.org
[11] http://www.ler-user.com

6 Conclusions and Future Work

In this paper we have proposed a general framework for distributed ontology systems, which aims to address issues arising from large ontologies. The framework suggests dividing a large ontology into fragments and distributing them over network. By doing so, the computation of queries posed over the large ontology can be handled in a distributed way. Distributed ontology systems also provide collaborative ontology development by allowing users to edit the ontology via any node. We described approaches for partitioning ontology and discussed query processing and collaborative development in such cases.

Our future work aims to design and implement algorithms for integrating the query answers from different nodes. As we have argued, one advantage of distributed ontology systems is distributed reasoning. However, the results from different nodes need to be combined on the master node in a manner such that the final result returned to the user is sound and complete. Another aim is to handle concurrency, which occurs when an ontology fragment is simultaneously involved in query processing and ontology editing.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi and P. F. Patel-Schneider, editors. The description logic handbook: theory, implementation, and applications. *Cambridge University Press*, 2003.
2. F. Baader, C. Lutz, and B. Suntisrivaraporn. CELa polynomial-time reasoner for life science ontologies. *In U. Furbach and N. Shankar, editors, Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of Lecture Notes in Artificial Intelligence, pages 287C291. Springer-Verlag, 2006.
3. D. Calvanese, G. Giacomo, and M. Lenzerini. Ontology of integration and integration of ontologies. In *Description Logic Workshop (DL 2001)*, pages 10C19, 2001.
4. D. Calvanese, G. D. Giacomo and M. Lenzerini. A framework for ontology integration. In *Isabel Cruz, Stefan Decker, Jérome Euzenat, and Deborah McGuinness, editors, The Emerging Semantic Web _ Selected Papers from the First Semantic Web Working Symposium*, pages 201-214. IOS Press, 2002.
5. D. Calvaese, G. De Giacomo, M. Lenzerini and M. Y. Vardi. View-based query processing and constraint satisfaction. In Proc. of the 15th IEEE Symp. on Logic in Computer Science(LICS2000), pages 361-371,2000.
6. T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. J. *of Intelligent and Cooperative Information Systems*, 2(4):375-398, 1993.
7. L. Serafini and A. Tamilin. DRAGO: Distributed Reasoning Architecture for the Semantic Web. *In Proceeds of the Second European Semantic Web Conference (ESWC'05)*, Springer-Verlag, 2005.
   http://sra.itc.it/projects/drago.
8. L. Serafini and A. Tamilin. Distributed Instance Retrieval in Heterogeneous Ontologies. In *Proc. of the Second Italian Semantic Web Workshop Semantic Web Applications and Perspectives (SWAP'05), CEUR Workshop Proceedings*, ISSN 1613-0073, 2005.

9. P. Doran, V. Tamma and L. Iannone. Ontology Module Extraction for Ontology Reuse: An Ontology Engineering Perspective. In *Proceedings of the 2007 ACM CIKM International Conference on Information and Knowledge Management (CIKM)*, Lisbon, Portugal, 2007.
10. B. C. Grau, I. Horrocks, Y. Kazakov and U. Sattler. A logical framework for mudularity of ontologies. In *IJCAI-07, Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, Hyderabad, India, January 2007*, pp.298-304. AAAI.
11. B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular reuse of ontologies: theory and practice. In *Journal of Artificial Intelligence Research*, 31:273-318, 2008.
12. P. Haase and B. Motik. A mapping system for the integration of OWL-DL ontologies. In *Axel Hahn, Sven Abels, Liane Haak, IHIS 05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pp. 9-16. ACM Press, November 2005.
13. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Journal of Web Semantics*, 1(4), 2004.
14. J. Hartmann, Y. Sure, P. Haase, R. Palma, and M. C. Surez-Figueroa. OMV C ontology metadata vocabulary. In *C. Welty, editor, ISWC 2005 Workshop on Ontology Patterns for the Semantic Web*, NOV 2005.
15. P. Haase, Y. Wang. A decentralized infrastructure for query answering over distributed ontologies. In *Proceedings of The 22nd Annual ACM Symposium on Applied Computing (SAC)*. ACM Press, Seoul, Korea, March 2007.
16. N. F. Noy. Semantic integration: a survey Of ontology-based approaches. *SIGMOD Record, Special Issue on Semantic Integration*, 33, 4. Published in 2004.
17. V. Komulainen, A. Valo and E. Hyvönen: A Collaborative Ontology Development and Service Framework ONKI. *Proceeding of ESWC*, 2005.
18. J. Seidenberg and A. Rector. The state of multi-user ontology engineering. In *proceeds of WoMo2007 workshop(at K-CAP2007)*.
19. S. Schulz, B. Suntisrivaraporn and F. Baader. SNOMED CT's Problem List: Ontologists' and Logicians' Therapy Suggestions. In *Proceedings of The Medinfo 2007 Congress, Studies in Health Technology and Informatics (SHTI-series)*. IOS Press, 2007.
20. L. Serafini and A. Tamilin. Local Tableaux for Reasoning in Distributed Description Logics. In *V. Haarslev and R. Moeller, editors, Proceedings of the 2004 Int. Workshop on Description Logics (DL2004)*, pages 100-109, CEUR-WS, 2004.