Querying Distributed Data through Distributed Ontologies: a Simple but Scalable Approach

François Goasdoué and Marie-Christine Rousset

CNRS - University of Paris Sud (LRI) & INRIA (Futurs)

LRI, Building 490, 91405, Orsay Cedex, France

{fg,mcr}@lri.fr

Abstract

In this paper, we define a simple but scalable framework for peer-to-peer data sharing systems, in which the problem of answering queries over a network of semantically related peers is always decidable. Our approach is characterized by a simple class-based language for defining peer schemas as hierarchies of atomic classes, and mappings as inclusions of logical combinations of atomic classes. We provide an anytime and incremental method for computing *all* the certain answers to a query posed to a given peer such that the answers are ordered from the ones involving peers close to the queried peer to the ones involving more distant peers.

1 Introduction

The Semantic Web [4] envisions a world-wide distributed architecture where data and computational resources will easily inter-operate to coordinate complex tasks such as answering queries or global computing. Semantic marking up of web resources using *ontologies* is expected to provide the necessary glue for making this vision work. The de-centralized nature of the Web makes inevitable that communities of users or software developers will use their own ontologies to describe their data or services. In this vision of the Semantic Web based on distributed ontologies, the key point is the mediation between data, services and users, using mappings between ontologies. The problem of schema mediation in a peer data management system (PDMS) has been investigated very recently in [8], where mappings between relational schemas are expressed using a powerful relational formalism. It is shown that in this setting, query answering is undecidable except if some restrictions are imposed on the mappings and on the resulting topology of the PDMS.

In this paper, we define a simpler framework, in which the problem of answering queries over a network of semantically related peers is always decidable. Our approach is characterized by a simple class-based language for defining peer schemas as hierarchies of atomic classes, and mappings as inclusions of logical combinations of atomic classes. An important point, following [9], is that those mappings may involve auxiliary classes of *helper* ontologies. Those auxiliary classes have a twofold usefulness: they serve for modeling overlaps between classes of different peer ontologies ; they also serve for reconciling classes of different ontologies by saying that they are (possibly disjoint) subclasses of some class. The feasibility and scalability of our approach is based on a propositional encoding which guarantees that there is a finite set of maximal rewritings for queries posed to the PDMS, and which characterizes them as prime implicants w.r.t a propositional theory.

The paper is organized as follows. Section 2 introduces our setting through an illustrate example, while Section 3 provides the underlying formal definitions and the related query answering problem, for which we distinguish *certain answers* from *potential answers*. In Section 4, we show the central role of rewritings to compute all the certain answers, and we exhibit the propositional encoding which is the basis of the incremental method described in Section 5. We conclude with a short discussion in Section 6.

2 Illustrative example

Suppose that Figure 1 sketches class hierarchies forming schemas (ontologies) of three distinct file servers storing teaching documents.

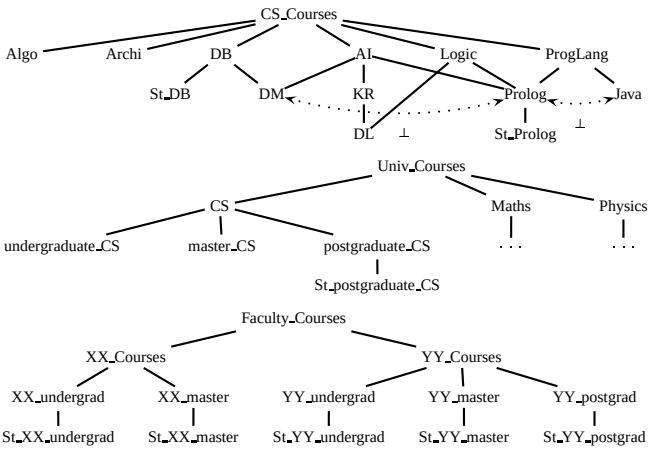

Figure 1: Three class hierarchies for teaching documents

The stored data are indicated by *extensional classes* whose name starts by St_. For instance, St_Prolog is a subclass of Prolog indicating that the server CS_Courses locally stores

courses on Prolog: their identifiers are the instances of the class St_Prolog.

The server CS_Courses is structured according to the different domains and sub-domains of computer science. For example, DM (standing for data mining) is declared as a subclass of both DB and AI, and disjoint from Prolog. Only courses on Prolog and DB (denoted by the extensional classes St_Prolog and St_DB) are stored within this server.

The University central server (Univ_Courses) structures the courses that it makes accessible according to their main subjects and then their levels. Only postgraduate courses on computer science are stored within this server.

Finally, the Faculty server (Faculty_Courses) groups teaching documents of Faculty members and hierarchically structures them according to the names of the teachers, and the levels of the corresponding courses. YY and XX have put all their courses available on that server.

There exists several correspondences between those three ontologies: Figure 2 shows possible logical mappings between ontologies of Figure 1.

$$\mathsf{XX\_undergrad} \sqsubseteq (\mathsf{Java} \sqcup \mathsf{Archi}) \sqcap \mathsf{undergraduate\_CS}$$
$$\mathsf{XX\_master} \sqsubseteq \mathsf{DL} \sqcap \neg\mathsf{DM}$$
$$\mathsf{YY\_Courses} \sqsubseteq \mathsf{CS\_Courses} \sqcap \neg\mathsf{Archi}$$
$$\mathsf{YY\_undergrad} \sqsubseteq (\mathsf{ProgLang} \sqcup \mathsf{Algo}) \sqcap \mathsf{undergraduate\_CS}$$
$$\mathsf{YY\_master} \sqcap \mathsf{YY\_postgrad} \sqsubseteq \mathsf{AI} \sqcup \mathsf{Logic}$$
$$\mathsf{Overlap1} \sqsubseteq \mathsf{YY\_postgrad} \sqcap \mathsf{KR}$$

Figure 2: Logical mappings between ontologies of Figure 1

For instance, the first mapping expresses that the undergraduate courses taught by XX are either undergraduate courses about java or architecture. The fifth mapping says that the common courses taught by YY at master and postgraduate levels deal with AI or Logic. The last mapping using an auxiliary class (Overlap1) states that there is an overlapping between the classes YY_postgrad and KR, in order to say that part of the YY's postgraduate courses taught (but not necessary all of them) deal with knowledge representation.

A query is a logical composition of classes of a given ontology, expressing which instances of which classes the user is interested in. For example, the query $Q_{\mathsf{CS\_Courses}} : \mathsf{AI} \sqcap \neg\mathsf{DM}$, posed in terms of the ontology CS_Courses, expresses that the user searches for courses on artificial intelligence which do no deal with data mining. The important point is that the answers can be inferred from the instances of the stored classes within the server CS_Courses, or from the instances of classes that are stored in the two other servers. The answers of $Q_{\mathsf{CS\_Courses}}$ that can be obtained locally (i.e., from the instances of the stored classes of the server CS_Courses) are courses about Prolog (the instances of St_Prolog). However, there also exists other answers for the query, which are stored in other servers, and which can be inferred from the mappings. In particular, the mapping: $\mathsf{XX\_master} \sqsubseteq \mathsf{DL} \sqcap \neg\mathsf{DM}$, and the fact that it can be inferred from the ontology rooted in CS_Courses that DL is a subclass of AI, allow us to infer that the instances of St_XX_master are also answers for the query $Q_{\mathsf{CS\_Courses}}$.

All those answers are *certain answers*: they are guaranteed to be in the answer set of the query. In some cases, in particular when there does not exist (enough) certain answers, it may be useful to provide *potential answers*. Potential answers are not guaranteed to belong to the answer set of the original query but to an overlapping set. Consider for example the mapping: $\mathsf{Overlap1} \sqsubseteq \mathsf{YY\_postgrad} \sqcap \mathsf{KR}$. Since KR is a subclass of AI, Overlap1 is a specialization of AI. The instances of the class YY_postgrad (stored in the extensional class St_YY_postgrad), which is a generalization of the specialization Overlap1 of the query AI, are considered as potential answers to AI.

3 Problem definition

First, we define the distributed data model of the peer data management systems (PDMS) that we deal with. Then, we state the query answering problem that we consider, distinguishing certain from potential answers.

Ontologies: syntax and semantics The ontologies that we deal with are very simple: they model hierarchies of *intentional classes* in the form of inclusion and disjointness statements between names of atomic classes. An inclusion statement is of the form: $A \sqsubseteq B$. A disjointness statement is of the form: $A \sqcap B \sqsubseteq \bot$.

For example, the ontology rooted in CS_Courses in Figure 1 is defined in Figure 3.

$\mathsf{ProgLang} \sqsubseteq \mathsf{CS\_Courses}$	$\mathsf{Algo} \sqsubseteq \mathsf{CS\_Courses}$
$\mathsf{Logic} \sqsubseteq \mathsf{CS\_Courses}$	$\mathsf{AI} \sqsubseteq \mathsf{CS\_Courses}$
$\mathsf{DB} \sqsubseteq \mathsf{CS\_Courses}$	$\mathsf{Archi} \sqsubseteq \mathsf{CS\_Courses}$
$\mathsf{Prolog} \sqsubseteq \mathsf{ProgLang}$	$\mathsf{Prolog} \sqsubseteq \mathsf{AI}$
$\mathsf{Prolog} \sqsubseteq \mathsf{Logic}$	$\mathsf{Java} \sqsubseteq \mathsf{ProgLang}$
$\mathsf{DM} \sqsubseteq \mathsf{AI}$	$\mathsf{DM} \sqsubseteq \mathsf{DB}$
$\mathsf{KR} \sqsubseteq \mathsf{AI}$	$\mathsf{DL} \sqsubseteq \mathsf{KR}$
$\mathsf{DL} \sqsubseteq \mathsf{Logic}$	$\mathsf{Java} \sqcap \mathsf{Prolog} \sqsubseteq \bot$
$\mathsf{DM} \sqcap \mathsf{Prolog} \sqsubseteq \bot$	

Figure 3: Statements defining the CS_Courses ontology

The semantics is defined in terms of *interpretations*. An interpretation I is a pair $(\Delta^I, .^I)$ where Δ^I is a non-empty set called the domain of interpretation of I, and $.^I$ is an interpretation function, which assigns a subset A^I of Δ^I to every atomic class A. An interpretation I is a *model* of an ontology O iff:

- for every inclusion $A \sqsubseteq B$ in O: $A^I \subseteq B^I$, and
- for every disjointness $A \sqcap B \sqsubseteq \bot$ in O: $A^I \cap B^I = \emptyset$.

An ontology O is *satisfiable* iff it has a model.

Storage description: syntax and semantics The storage description of a peer whose local schema is defined by the ontology O is a set of declarations of extensional classes. The declaration of an extensional class St_A consists of:

- an inclusion statement relating St_A to one or more classes of O,
- an extension, denoted $\mathcal{E}(St\_A)$, which is a set of distinct constants representing data identifiers, that are instances of the St_A.

The simplest inclusion statements are of the form: $St\_A \sqsubseteq A$. They express that the corresponding peer locally stores a subset of the class A. The general form of an inclusion statement of an extensional class is: $St\_A \sqsubseteq Q$, where Q can

be a logical combination of class literals. A class literal is either a class name of O or the negation of a class name of O.

Given an interpretation I, we extend its interpretation function to the extensional classes and to the constants appearing in their extensions: each constant a is interpreted as an element a^I of the domain of interpretation Δ^I. The interpretation of a logical combination of class literals is inductively defined as follows:

- $(\neg A)^I = \Delta^I \setminus A^I$
- $(C \sqcap D)^I = C^I \cap D^I$
- $(C \sqcup D)^I = C^I \cup D^I$

An interpretation I is a model of a storage description iff for each assertional class defined by its extension $\mathcal{E}(St\_A)$, and by the inclusion $St\_A \sqsubseteq Q$:

- for each $a \in \mathcal{E}(St\_A)$, $a^I \in St\_A^I$,
- $St\_A^I \subseteq Q^I$.

Storage descriptions correspond to *sound views* in the general setting of information integration defined in [5].

Mappings: syntax and semantics A mapping has the form of an inclusion statement $Q_1 \sqsubseteq Q_2$, where Q_1 and Q_2 are logical combinations of class literals involving intentional classes only, coming from at least two different ontologies.

We distinguish *overlap* mappings, which are of the form $Overlap_i \sqsubseteq A_{i1} \sqcap \cdots \sqcap A_{ik}$ such that $A_{i1}, \ldots, A_{ik}$ are intentional classes of distinct ontologies, and $Overlap_i$ is a fresh name of class. *Overlap* mappings serve for expressing that there exists an overlap between classes of different ontologies. Their role for computing potential answers for queries will be explained in Section 6.

Given an interpretation I, we extend its interpretation function to the classes $Overlap_i$ by assigning to them a non empty subset of the domain of interpretation.

An interpretation I is a model of a set of mappings $\mathcal{M}$ iff for every mapping $Q_1 \sqsubseteq Q_2$ in $\mathcal{M}$, $Q_1^I \subseteq Q_2^I$.

PDMS: schema and extension We denote by $\mathcal{O}$ (respectively $\mathcal{O}_s$) the union of the definitions of intentional classes (respectively the union of the definitions of intentional *and* extensional classes) of the distributed ontologies of a PDMS $\mathcal{P}$. Without loss of generality we assume that class names are unique to each peer, and that the names of extensional classes are distinct from those of intentional classes. The schema of a PDMS is defined by $\mathcal{O}_s$ and by a set $\mathcal{M}$ of mappings. It is said *satisfiable* iff there exists a model of $\mathcal{O}_s$ and $\mathcal{M}$.

Given a storage description representing the distributed data stored within $\mathcal{P}$, the extension of $\mathcal{P}$ is the union $\mathcal{E}$ of the extensions of the extensional classes in $\mathcal{O}_s$. We will use the notation $\mathcal{P} = \langle \mathcal{O}_s, \mathcal{M}, \mathcal{E} \rangle$ to denote the three components of a PDMS $\mathcal{P}$.

The *neighborhood graph* accounts for the connection between the different peers within a given PDMS induced by the mappings.

Definition 1 (Neighborhood graph) *Let $\mathcal{P}$ be a PDMS. Its* neighborhood graph *is a graph (V,E) where V is the set of peers of $\mathcal{P}$, and (P_i, P_j) is an edge of E if there exists a mapping involving classes of the ontologies of P_i and P_j.*

The *peer distance* between two peers P and P' of a PDMS $\mathcal{P}$ is the length of the shortest path between P and P' in the neighborhood graph of $\mathcal{P}$.

Query answering The queries that we consider are logical combinations of intentional class literals of a given peer ontology. The following definition is the logical counterpart of the database definition of certain answers [1; 5; 9].

Definition 2 (Certain answers) *Let Q be a query over a PDMS $\mathcal{P} = \langle \mathcal{O}_s, \mathcal{M}, \mathcal{E} \rangle$. a is a* certain answer *of Q iff $a^I \in Q^I$ for every model I of $\mathcal{O}_s \cup \mathcal{M} \cup \mathcal{E}$.*

We now define the notion of potential answers (which distinguishes from the notion of possible answers, as defined in [5]). It relies on the notion of query generalization and specialization.

Definition 3 (Query generalization/specialization) *Given a PDMS $\mathcal{P} = \langle \mathcal{O}_s, \mathcal{M}, \mathcal{E} \rangle$, let Q and Q' be two logical combinations of class literals of $\mathcal{O}_s$. Q' is a* generalization *of Q (and Q is a* specialization *of Q') iff $\mathcal{O}_s \cup \mathcal{M} \cup \{Q\}$ is satisfiable and $\mathcal{O}_s, \mathcal{M}, Q \models Q'$.*

Definition 4 (Potential answers) *Let Q be a query over a PDMS $\mathcal{P} = \langle \mathcal{O}_s, \mathcal{M}, \mathcal{E} \rangle$. Let a be in $\mathcal{E}$. a is a* potential answer *of Q iff a is a certain answer of a generalization of Q or of one of its specializations.*

Given a PDMS $\mathcal{P}$ and a query Q, the query answering problem that we are interested in has two variants: (1) find all certain answers of Q ; (2) find potential answers of Q.

In general, finding all certain answers is a critical issue [8]. In our setting however, we are in a case where all the certain answers can be obtained using rewritings of the query (see Section 4). Given a query Q posed to some peer P, the important point is to be able to order the certain answers for Q that can be obtained from the other peers, according to their proximity to P (see Section 5). Finding potential answers can be useful if there is not (enough) certain answers. By definition, potential answers may be numerous. The problem is to focus on some kind of potential answers. We will briefly discuss this point in Section 6.

4 Query answering using rewritings

We first define the notion of (maximal) conjunctive rewriting of a query in our setting. We then show how conjunctive rewritings can be used to compute all the certain answers for the query. Finally, we characterize the maximal conjunctive rewritings of a query as prime implicants of the propositional encoding of the query w.r.t the propositional theory encoding the mappings and the ontologies.

Definition 5 (Conjunctive rewriting of a query) *Let Q be a query. Let Q_e be a conjunction of extensional classes. Q_e is a* conjunctive rewriting *of Q iff Q_e is a specialization of Q. It is* maximal *iff there does not exist a strict generalization of Q_e which is a rewriting of Q.*

Evaluating conjunctive rewritings of a query Q provides certain answers of Q. The evaluation of conjunctive rewriting $Q_e : St\_A_1 \sqcap \cdots \sqcap St\_A_n$ is direct:

$$Eval(Q_e) = \mathcal{E}(St\_A_1) \cap \cdots \cap \mathcal{E}(St\_A_n).$$

Most importantly, it has been shown in [6; 7] that when a query has a *finite number of maximal conjunctive rewritings*, then the complete set of its answers can be obtained (in polynomial data complexity) as the union of the answer sets of its rewritings. We go through a *propositional encoding* to show that in our setting every query has a finite number of maximal conjunctive rewritings.

Propositional encoding The propositional encoding $Prop(Q)$ of a query Q is a propositional formula using the names of atomic classes as propositional variables, which is inductively defined as follows:

- $Prop(A) = A$, if A is an atomic class;
- $Prop(\neg A) = \neg A$, if A is an atomic class;
- $Prop(Q_1 \sqcap Q_2) = Prop(Q_1) \wedge Prop(Q_2)$;
- $Prop(Q_1 \sqcup Q_2) = Prop(Q_1) \vee Prop(Q_2)$.

The propositional encoding of a PDMS $\mathcal{P} = \langle \mathcal{O}_s, \mathcal{M}, \mathcal{E} \rangle$ is the set of propositional formulas $Prop(s)$ obtained as follows from each statement s in $\mathcal{O}_s$ and $\mathcal{M}$:

- $Prop(A \sqsubseteq Q) = A \Rightarrow Prop(Q)$;
- $Prop(A \sqcap B \sqsubseteq \bot) = \neg A \vee \neg B$;
- $Prop(Q_1 \sqsubseteq Q_2) = Prop(Q_1) \Rightarrow Prop(Q_2)$.

The following proposition shows that propositional encoding transfers the logical definitions and properties previously introduced for classes. It also provides a propositional characterization of maximal conjunctive rewritings of a query as *prime implicants w.r.t a theory* [10].

Proposition 1 (Propositional transfer) *Let $\mathcal{P}$ be a PDMS. Let $\mathcal{O}_{prop} \cup \mathcal{M}_{prop}$ be the propositional encoding of its schema. Let V_e be the set of names of its extensional classes.*

- $\mathcal{P}$'s schema is satisfiable iff $\mathcal{O}_{prop} \cup \mathcal{M}_{prop}$ is satisfiable.

- Q_e is a maximal conjunctive rewriting of Q iff $Prop(Q_e)$ is a prime implicant of $Prop(Q)$ w.r.t the theory $\mathcal{O}_{prop} \cup \mathcal{M}_{prop}$ *among the implicants that are conjunctions of propositional variables of V_e.*

As a result, we can use any SAT algorithm for checking satisfiability of PDMS schemas. Most importantly, it gives us a way to compute *all* the certain answers of a query by rewriting.

From now on, for simplicity purpose, we use the propositional notation for the queries, the ontologies, the mappings and the rewritings. We suppose that all the propositional formulas that we consider are in clausal form. We suppose that PDMS satisfiability has been checked, and we focus on the computation of the maximal rewritings of an *atomic* query. Note that the maximal rewritings of a conjunctive query can be obtained by combining the maximal rewritings of its atomic conjuncts.

In the following section, we give the sketch of an anytime and incremental method for computing maximal conjunctive rewritings of atomic queries. It follows an order induced by the neighborhood graph such that the maximal rewritings (and thus the answers) involving peers close to the interrogated peer are obtained first.

5 An anytime rewriting algorithm

Proposition 1 characterizes maximal rewritings as conjunctive prime implicants of the (propositional encoding of the) query w.r.t the propositional theory encoding the ontologies and the mappings of the PDMS. The following property shows that the problem of finding prime implicants can be reduced to that of finding *prime implicates*.

Proposition 2 (Connection prime implicants / implicates) *Let Q be a query and T a propositional theory. The conjunctive prime implicants of Q w.r.t T are the negation of the clausal prime implicates of $\neg Q$ w.r.t T.*

We reuse a graph-based technique [2; 3] for computing prime implicates within propositional theories partitioned into sub-theories. A partitioned theory induces a graph called the *intersection graph*: each node represents a sub-theory of the partitioning ; two nodes are linked with an edge if they share propositional variables ; an edge is labeled with these shared variables.

The *forward message-passing algorithm* MP described in [2] exploits the partitioning to provide an efficient consequence-finding algorithm. Consequence finding is done in parallel in each individual sub-theory using any complete resolution strategy. The transfer of formulas between sub-theories is controlled by the labels of the edges of the intersection graph: the logical consequences found within an individual sub-theory are sent as messages to another individual sub-theory (and are added to the set of its formulas) *only if* they involve propositional variables that are shared between those two sub-theories. Let us illustrate the message-passing behavior of this algorithm on the intersection graph of Figure 4.

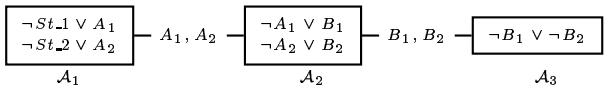

Figure 4: Example of a partitioning and its intersection graph

Suppose that you are interested in finding all the clausal prime implicates using the variables $\mathsf{St\_j}$, i.e., the prime implicates that can be obtained within the sub-theory $\mathcal{A}_1$. The algorithm starts sending messages from the most distant sub-theory from $\mathcal{A}_1$: $\mathcal{A}_3$. The formula $\neg \mathsf{B}_1 \vee \neg \mathsf{B}_2$ is transmitted from $\mathcal{A}_3$ to $\mathcal{A}_2$. Within $\mathcal{A}_2$, it is inferred by application of resolution rules the new formulas: $\neg \mathsf{A}_1 \vee \neg \mathsf{B}_2$, $\neg \mathsf{A}_2 \vee \neg \mathsf{B}_1$, $\neg \mathsf{A}_1 \vee \neg \mathsf{A}_2$. Now, only $\neg \mathsf{A}_1 \vee \neg \mathsf{A}_2$ is transmitted to the sub-theory $\mathcal{A}_1$, and we finally obtain the prime implicate: $\neg \mathsf{St}_1 \vee \neg \mathsf{St}_2$.

In order to be complete, this forward message-passing algorithm must apply to an intersection graph without cycle. The point is that any intersection graph can be polynomially transformed into a cycle-free graph with enlarged labels such that applying the forward message-passing algorithm to this acyclic graph is guaranteed to be complete. The BREAK-CYCLES algorithm described in [2] performs the appropriate transformation.

We describe our partitioning in Section 5.1. It is independent of the queries and thus can be done at *compile time*. It follows the natural partition of the ontologies over the different peers but it is also induced by the target prime implicates: we group all the formulas involving names of ex-

tensional classes as propositional variables into a single sub-theory called the *warehouse*. In Section 5.2 and Section 5.3 we sketch the way we have used the BREAK-CYCLES and MP algorithms of [2] to obtain an *anytime* and *incremental* algorithm for computing the clausal prime implicates of the negation of an atomic query. We encapsulate the MP algorithm [2] in an iterative loop: at the first step, MP is applied to the cycle-free intersection graph of the partitioning restricted to the warehouse and the sub-theory corresponding to the ontology of the queried peer ; each iteration takes into account intersection graphs corresponding to the warehouse, the sub-theory corresponding to the ontology of the queried peer as well as the ontologies (and the related mappings) of peers within an increasing peer distance to the queried peer. It is important to note that this peer distance is not the distance of the sub-theories within the intersection graph, but the distance of the corresponding peers within the PDMS neighborhood graph. In order to avoid applying BREAK-CYCLES at each iterative step of the prime implicates computation, we break the cycles at *compile time*. At query time, we just have to load the cycle-free intersection graphs corresponding to the queried peer as input to the incremental forward message-passing algorithm.

5.1 Our partitioning

It is guided by the natural distribution of the PDMS but groups all the definitions of extensional classes together:

- we group in a sub-theory the formulas defining the intentional classes of a same ontology;

- we group in a sub-theory the mapping formulas that are associated with the same subset of ontologies [1];

- all the definitions of extensional classes in the ontologies are grouped into an individual sub-theory: the *warehouse*.

Consider the PDMS $\langle O_s^1 \cup O_s^2 \cup O_s^3, \mathcal{M}, \mathcal{E} \rangle$ such that:

O_s^1 : $\mathsf{A} \sqsubseteq \mathsf{O}_1, \mathsf{St\_A} \sqsubseteq \mathsf{A}$
O_s^2 : $\mathsf{B} \sqsubseteq \mathsf{O}_2, \mathsf{St\_B} \sqsubseteq \mathsf{B}, \mathsf{B}' \sqsubseteq \mathsf{O}_2$
O_s^3 : $\mathsf{C} \sqsubseteq \mathsf{O}_3, \mathsf{St\_C} \sqsubseteq \mathsf{C}$
$\mathcal{M}$: $\mathsf{B} \sqcap \mathsf{B}' \sqsubseteq \mathsf{A}, \mathsf{C} \sqsubseteq \mathsf{B}'$

The intersection graph resulting of its partitioning is given in Figure 5.

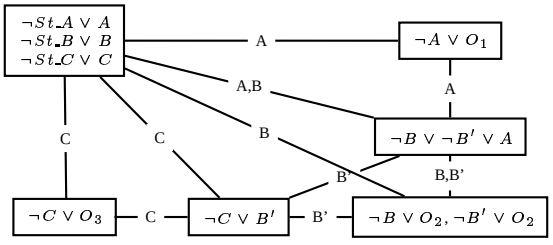

Figure 5: Intersection graph resulting from the partitioning

5.2 Breaking cycles

The following algorithm is applied at *compile time* for each peer P of the PDMS.

[1] In the general case, a mapping may involve classes of more than two distinct ontologies

Let n be the peer distance between P and the most distant peer from P within the PDMS
For $k = 0$ to n,

- let T_k be the intersection graph restricted to the warehouse, the sub-theories encoding the ontologies of P and the ontologies (and the associated mappings) of the peers whose peer distance from P is less than k
- apply BREAK-CYCLES to T_k

The results $T_0, \ldots, T_n$ computed by this algorithm are the cycle-free intersection graphs to which the forward message-passing algorithm will be iteratively applied at query time if P is the interrogated peer. Figure 6 illustrates the successive results T_1 and T_2 obtained for the peer corresponding to O_s^1 in the above PDMS (the removed edges for breaking cycles are indicated by dotted lines).

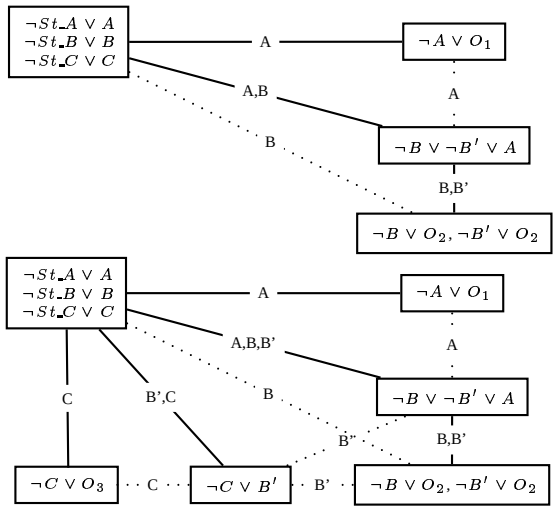

Figure 6: Cycle-free intersection graphs T_1 and T_2 for O_s^1

5.3 Ordered computing of implicates

The following algorithm computes the clausal implicates of the negation of an atomic query Q posed to a given peer P [2].

Let $T_0, T_1, \ldots, T_n$ be the successive cycle-free intersection graphs computed for P at compile time (Section 5.2)
For $k = 0$ to n,

- add $\neg Q$ to the sub-theory of T_k encoding the ontology of P
- apply MP to T_k to compute the implicates of distance $\leq k$

For example, consider that the query A is posed to the peer corresponding to the ontology O_s^1 in the above PDMS.

Figure 7 illustrates the result of our algorithm at step $k = 0$.

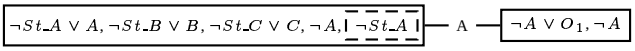

Figure 7: Local implicates (within a peer distance of 0)

$\neg\mathsf{A}$, which has been added to the sub-theory encoding O_s^1, is sent to the warehouse. A resolution is then possible

[2] We consider that the implicates of the theory alone have been computed at compile time

within the warehouse, which produces the implicate $\neg$St_A: its negation corresponds to the only rewriting that can be produced within the interrogated peer (peer distance of 0).

The application of MP to T_1 (first cycle-free graph in Figure 6) infers no new formula.

Figure 8 illustrates the result obtained at the last step ($k = 2$) of our algorithm. It corresponds to the application of MP to T_2 (the last cycle-free graph in Figure 6): the clauses $\neg B \vee \neg B' \vee A$ and $\neg C \vee B'$ are transmitted to the warehouse, thus leading to the computation in the warehouse of the new implicate $\neg$St_B $\vee \neg$St_C for $\neg A$. Its negation is the conjunctive rewriting St_B $\wedge$ St_C, whose evaluation will produce the certain answers obtainable within a peer distance of 2.

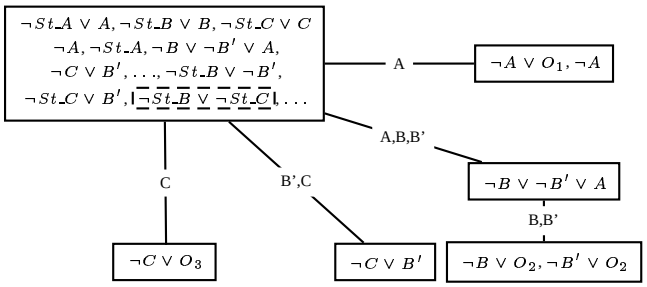

Figure 8: Implicates within a peer distance of 2

6 Discussion

Existing information integration systems are centralized systems of mediation between users and distributed data, which exploit mappings between a single mediated schema and schemas of data sources. For scaling up to the Web, this centralized approach of mediation is probably not flexible enough, and distributed systems of mediation are more appropriate. The approach that we have presented in this paper is an instance of the general PDMS architecture introduced in [9; 8], for which we guarantee decidability of query answering independently of the topology of the PDMS.

We have sketched an incremental and anytime method to compute and rank the certain answers for users's queries. It is based on the use of an existing graph-based technique ([2]) for reasoning in partitioned propositional theories. For space limitation, we have not presented the optimization of the method described in Section 5.2 for breaking cycles which enables to store for each peer a single cycle-free intersection graph T_n, instead of storing the successive cycle-free intersection graphs $T_1, \ldots, T_n$. This optimization requires an adaptation of the BREAK-CYCLES algorithm described in [2] to guarantee that the results obtained by the iterative forward message-passing algorithm on restrictions of the single cycle-free intersection graph that is stored at compile time are the same as if it applied BREAK-CYCLE at each iterative step. At query time, the method is anytime because it can stop at each step and then provides the certain answers that have been computed until this step. It is incremental in the sense that the computation done in order to produce implicates from some peers is reused to compute answers from more distant peers. The scalability of the approach relies on the propositional encoding: adding a new peer consists in updating the propositional theory (and the resulting intersection graph) by adding the logical formulas corresponding to the new ontology and to the mappings between some new classes and some semantically related existing classes in other ontologies.

As for potential answers (Definition 4) the problem is to limit the generalizations (of the original query or of one of its specializations) for which computing certain answers. Our strategy, which we just have enough space to mention here, is to focus on *overlapping queries*. An overlapping query of an atomic query A is a query A_i such that there exists an overlap mapping of the form: Overlap $\sqsubseteq A_1 \sqcap \ldots A_i \sqcap \ldots A_j \sqcap \ldots$, where A_j is a subclass specializing A. The point is that overlapping queries and their peer distance to the original query A can be easily detected at query time: the production of a unit clause $\neg$Overlap into a sub-theory (encoding a set of mappings) results from a resolution between a clause $\neg$Overlap $\vee A_j$ with a literal $\neg A_j$ which is necessarily an implicate of the negation of the atomic query. The overlapping queries that can be found within that sub-theory are those A_i such that $\neg$Overlap $\vee A_i$ is a formula of the sub-theory. Such a formula necessarily exists in the sub-theory, as the result of putting in clausal form the propositional encoding of the corresponding overlap mapping.

References

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of PODS*, 1998.

[2] E. Amir and S. A. McIlraith. Partition-based logical reasoning. In *Proceedings of KR*, 2000.

[3] E. Amir and S. A. McIlraith. Theorem proving with structured theories. In *Proceedings of IJCAI*, 2001.

[4] T. Berners-Lee, J. Hendler, and O.Lassila. The semantic web. *Scientific American, 279*, 2001.

[5] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Answering regular path queries using views. In *Proceedings of ICDE*, 2000.

[6] F. Goasdoué. *Réécriture de requêtes en termes de vues dans CARIN et intégration d'informations*. PhD thesis, Université Paris Sud XI - Orsay, 2001.

[7] F. Goasdoué and M.-C. Rousset. Answering queries using views: a krdb perspective for the semantic web. *Technical report, submitted for publication*, 2003.

[8] A. Halevy, Z.Ives, D.Suciu, and I.Tatarinov. Schema mediation in peer data management systems. In *Proceedings of ICDE*, 2003.

[9] J. Madhavan, P. Bernstein, P. Domingos, and A. Halevy. Representing and reasoning about mappings between domain models. In *Proceedings of IJCAI*, 2001.

[10] P. Marquis. Knowledge compilation using theory prime implicates. In *Proceedings of IJCAI*, 1995.