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

François Goasdoué and Marie-Christine Rousset, *Université Paris-Sud*

This framework for peer-to-peer data-sharing systems allows efficient query answering over a network of semantically related peers. A simple class-based language appropriate for practical applications defines peer schemas as hierarchies of atomic classes and mappings as inclusions of logical combinations of atomic classes.

The Semantic Web[1] envisions a worldwide distributed architecture where data and computational resources will easily interoperate to coordinate complex tasks, such as query answering and distributed computing. The semantic marking-up of Web resources using *ontologies* is expected to provide the necessary glue for making this vision work. Ontologies are structured vocabularies shared by communities of users who are not necessarily knowledge engineers or logicians. So, the ontologies must be simple. This doesn't mean that they lack a well-defined formal semantics: because they are simple, their formal semantics can fit nicely with their intuitive meaning. Indeed, the ontologies that type Web data must have a formal semantics to enable precise and rigorous characterization of operations on those data. For example, we can rigorously define (and thus prove) a query-answering algorithm's soundness and completeness only with respect to a given formal semantics of the data queried and the language used to do so.

Because of the Web's decentralized nature, communities of users will inevitably use their own ontologies to describe their data or services. In this vision of the Semantic Web, mediation among data, services, and users is key. Query answering in peer data management systems is in general undecidable (not feasible) unless you impose restrictions on the mappings and on the PDMS's resulting topology.[2]

This article defines a simple framework appropriate for practical applications, in which the problem of answering queries over a network of semantically related peers is always decidable. Although our framework is based on (propositional) logic, it's simple enough for nonlogicians (which most application developers are) to use. Our approach also copes with many information integration application requirements: It offers a class-based language for defining peer schemas as hierarchies of (possibly disjoint) atomic classes and mappings as inclusions of logical combinations of atomic classes. Because our algorithm guarantees both the soundness and completeness of the answers to users' queries, this approach suits information integration applications such as e-commerce where content providers expect PDMSs to propose all products that match a user's need.

Example

Suppose we have schemas (ontologies) for three distinct file servers that store teaching documents. Figure 1 shows the class hierarchies forming these ontologies.

Extensional classes, whose names start with **St_**, indicate stored data. For instance, **St_Prolog** is a subclass of the **Prolog** class, indicating that the CS_Courses server (see Figure 1a) locally stores courses on Prolog: their identifiers are instances of the class **St_Prolog**. We structured the CS_Courses server according to the different computer science domains and subdomains. For example, **DM** (for data mining) is declared a subclass of both **DB** and **AI** and disjoint from **Prolog**. The server stores only courses on Prolog and databases (denoted by the extensional classes **St_Prolog** and **St_DB**) here.

The university central server Univ_Courses (see Figure 1b) structures the courses that it makes accessible according to their main subjects and then their levels. This server stores only postgraduate computer science courses.

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

The Faculty_Courses server (see Figure 1c) groups the faculty members' teaching documents; it hierarchically structures them according to the teachers' names and the levels of the corresponding courses. Teachers YY and XX have put all their courses on that server.

Figure 2 shows the possible logical mappings between the three ontologies shown in Figure 1. For instance, the first mapping expresses that the undergraduate courses taught by XX are either about Java or architecture.

A *query* is a logical composition of the classes of a given ontology, expressing which instances of which classes the user is interested in. For example, the query $Q_{CS\_Courses}$: AI ⊓ ¬DM, posed in terms of the ontology CS_Courses, states that the user is searching for courses on artificial intelligence that don't deal with data mining. (See the "Symbols" sidebar for explanations of these symbols as well as others used in the article.) The important point is that we can infer answers from the instances of the classes stored in the server CS_Courses or in one of the other two servers. The answers for $Q_{CS\_Courses}$ that we can obtain locally (that is, from CS_Courses) are courses about Prolog (the instances of St_Prolog). However, other answers for the query are stored in other servers and can be inferred from the mappings. In particular, the mapping XX_master ⊑ DL ⊓ ¬ DM, and the fact that we can infer from the ontology rooted in CS_Courses that DL is a subclass of AI, lets us infer that the instances of St_XX_master are also answers to the query $Q_{CS\_Courses}$.

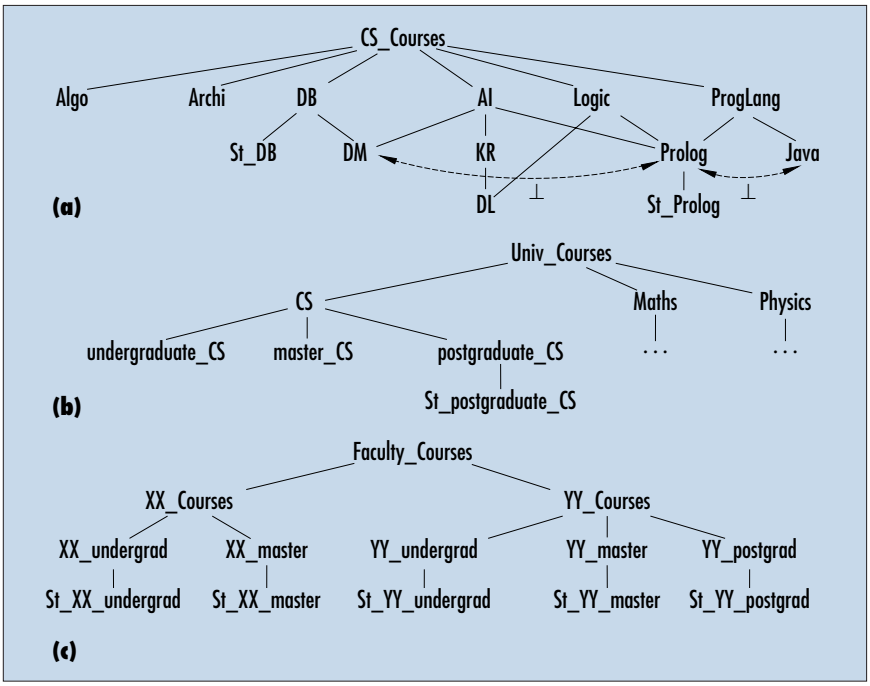


Figure 1. Three class hierarchies of teaching documents: (a) CS_Courses, (b) Univ_Courses, and (c) Faculty_Courses.

XX_undergrad ⊑ Java ⊔ Archi) ⊓ undergraduate_CS
XX_master ⊑ DL ⊓ ¬ DM
YY_Courses ⊑ CS_Courses ⊓ ¬ Archi
YY_undergrad ⊑ (ProgLang ⊔ Algo) ⊓ undergraduate_CS
YY_master ⊓ YY_postgrad ⊑ AI ⊔ Logic

Figure 2. Logical mappings between the ontologies in Figure 1.

ProgLang ⊑ CS_Courses
Logic ⊑ CS_Courses
DB ⊑ CS_Courses
Prolog ⊑ ProgLang
Prolog ⊑ Logic
DM ⊑ AI
KR ⊑ AI
DL ⊑ Logic
DM ⊓ Prolog ⊑ ⊥
Algo ⊑ CS_Courses
AI ⊑ CS_Courses
Archi ⊑ CS_Courses
Prolog ⊑ AI
Java ⊑ ProgLang
DM ⊑ DB
DL ⊑ KR
Java ⊓ Prolog ⊑ ⊥

Figure 3. Statements defining the CS_Courses ontology.

Problem definition

Before we present the query-answering problem addressed in this article, we must define the distributed data model of the PDMSs we are dealing with.

Syntax and semantics

Our ontologies are simple: they model hierarchies of *intentional classes* (classes that only organize knowledge on servers; they do not store data) in the form of inclusion (A ⊑ B) and disjointness (A ⊓ B ⊑ ⊥) statements between names of atomic classes. For example, Figure 3 defines the ontology rooted in CS_Courses in Figure 1.

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

For every inclusion A ⊑ B in O: $A^I \subseteq B^I$ (where ⊑ is inclusion between classes at knowledge level and ⊆ is inclusion between data/individuals of classes)
For every disjointness A ⊓ B ⊑ ⊥ in O: $A^I \cap B^I = \emptyset$.

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

The storage description of a peer whose local schema is defined by the ontology O is

Symbols

- ⊑ : Inclusion between classes at the knowledge level
- ⊆ : Inclusion between data/individuals of classes
- |= : Entails; that is, $\mathcal{O}_s$, $\mathcal{M}$, Q |= Q′ means O_s, $\mathcal{M}$, Q entails Q′, or Q′ is a logical consequence of O_s, $\mathcal{M}$, Q)
- {Q} : A set containing Q
- ∪ : Union of sets as arguments
- ∩ : Set intersection
- ⊓ : Conjunction between classes
- ¬ : Negation of classes

a set of declarations of extensional classes. The declaration of an extensional class $\mathsf{St\_A}$ consists of an inclusion statement relating $\mathsf{St\_A}$ to one or more classes of O and an extension, denoted $\mathcal{E}(\mathsf{ST\_A})$, which is a set of distinct constants representing data identifiers that are instances of the $\mathsf{St\_A}$. The simplest inclusion statements are of the form $\mathsf{St\_A} \sqsubseteq \mathsf{A}$, meaning that the corresponding peer stores a subset of the class A locally. The general form of an extensional class's inclusion statement is $\mathsf{St\_A} \sqsubseteq \mathsf{Q}$, 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 $\triangle^\mathsf{I}$. The interpretation of a logical combination of class literals is inductively defined as follows:

$$(\neg \mathsf{A})^\mathsf{I} = \triangle^\mathsf{I} \setminus \mathsf{A}^\mathsf{I}$$
$$(\mathsf{C} \sqcap \mathsf{D})^\mathsf{I} = \mathsf{C}^\mathsf{I} \cap \mathsf{D}^\mathsf{I}$$
$$(\mathsf{C} \sqcup \mathsf{D})^\mathsf{I} = \mathsf{C}^\mathsf{I} \cup \mathsf{D}^\mathsf{I}$$

An interpretation I is a model of a storage description iff for each assertional class defined by its extension (that is, for each $\mathsf{a} \in \mathcal{E}(\mathsf{St\_A})$, $\mathsf{a}^\mathsf{I} \in \mathsf{St\_A}^\mathsf{I}$) and by the inclusion $\mathsf{St\_A} \sqsubseteq \mathsf{Q}$, $\mathsf{St\_A}^\mathsf{I} \subseteq \mathsf{Q}^\mathsf{I}$. Storage descriptions correspond to *sound views* in the general setting of information integration.[3]

A *mapping* has the form of an inclusion statement $\mathsf{Q}_1 \sqsubseteq \mathsf{Q}_2$, where Q_1 and Q_2 are logical combinations of class literals involving intentional classes only, coming from at least two different ontologies.

An interpretation I is a model of a set of mappings $\mathcal{M}$ iff for every mapping $\mathsf{Q}_1 \sqsubseteq \mathsf{Q}_2$ in $\mathcal{M}$, $\mathsf{Q}_1^\mathsf{I} \subseteq \mathsf{Q}_2^\mathsf{I}$.

PDMS schema and extensions

We denote by $\mathcal{O}$ (or $\mathcal{O}_s$) the union of the definitions of intentional classes (or the union of the definitions of intentional and extensional classes, respectively) 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. $\mathcal{O}_s$ and a set $\mathcal{M}$ of mappings define the schema of a PDMS. It is said to be *satisfiable* iff there exists a model of $\mathcal{O}_s$ and $\mathcal{M}$.

Given a description representing the distributed data stored in $\mathcal{P}$, the extension of $\mathcal{P}$ is the union $\mathcal{E}$ of the extensions of the extensional classes in O_s. We 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* shows 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 a mapping exists 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.[3–5]

Definition 2. 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 an *answer* of Q iff $\mathsf{a}^\mathsf{I} \in \mathsf{Q}^\mathsf{I}$ for every model I of $\mathcal{O}_s \cup \mathcal{M} \cup \mathcal{E}$.

Our query-answering problem

Given a PDMS $\mathcal{P}$ and a query Q, our query-answering problem is to find all the answers of Q. Finding all answers is generally a critical issue.[2]

Query answering using rewriting

The notion of the (maximal) conjunctive rewriting of a query relies on query generalization and specialization.

Definition 3. Query generalization and 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 $\mathcal{O}_s \cup \mathcal{M} \cup \{\mathsf{Q}\}$ is satisfiable and $\mathcal{O}_s, \mathcal{M}, \mathsf{Q} \models \mathsf{Q}'$.

Definition 4. Conjunctive rewriting of a query: Let Q be a query and 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 that is a rewriting of Q.

Evaluating conjunctive rewritings of Q provides answers to Q. The evaluation of conjunctive rewriting Q_e: $\mathsf{St\_A}_1 \sqcap \ldots \sqcap \mathsf{St\_A}_n$ is direct:

$$\text{Eval } \mathsf{Q}_e = \mathcal{E}(\mathsf{St\_A}_1) \cap \ldots \cap \mathcal{E}(\mathsf{St\_A}_n).$$

Most importantly, when a query has a finite number of maximal conjunctive rewritings, we can obtain the complete set of its answers (in polynomial data complexity) as the union of the answer sets of its rewritings.[6,7]

Propositional encoding

The following *propositional encoding* of a PDMS is the first step in showing that in our setting, every query has a finite number of maximal conjunctive rewritings. The propositional encoding $\mathsf{Prop(Q)}$ of a query Q is a propositional formula using the names of atomic classes as propositional variables. We define it inductively as follows:

$$\mathsf{Prop(A)} = \mathsf{A} \quad \text{if } \mathsf{A} \text{ is an atomic class}$$
$$\mathsf{Prop}(\neg\mathsf{A}) = \neg\mathsf{A} \quad \text{if } \mathsf{A} \text{ is an atomic class}$$
$$\mathsf{Prop}(\mathsf{Q}_1 \sqcap \mathsf{Q}_2) = \mathsf{Prop}(\mathsf{Q}_1) \wedge \mathsf{Prop}(\mathsf{Q}_2)$$
$$\mathsf{Prop}(\mathsf{Q}_1 \sqcup \mathsf{Q}_2) = \mathsf{Prop}(\mathsf{Q}_1) \vee \mathsf{Prop}(\mathsf{Q}_2)$$

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

$$\mathsf{Prop}(\mathsf{A} \sqsubseteq \mathsf{Q}) = \mathsf{A} \Rightarrow \mathsf{Prop(Q)}$$
$$\mathsf{Prop}(\mathsf{A} \sqcap \mathsf{B} \sqsubseteq \bot) = \neg\mathsf{A} \vee \neg\mathsf{B}$$
$$\mathsf{Prop}(\mathsf{Q}_1 \sqsubseteq \mathsf{Q}_2) = \mathsf{Prop}(\mathsf{Q}_1) \Rightarrow \mathsf{Prop}(\mathsf{Q}_2)$$

Propositional transfer

The following proposition shows that propositional encoding transfers the logical definitions and properties previously introduced for classes to the propositional logic setting. It also provides a propositional characterization of maximal conjunctive rewritings of a query as *prime implicants* with regard to a theory.[8] (A prime implicant PI of a formula F wrt (with regard to) a theory T is such that PI $\models$ F wrt T (PI is an implicant of F wrt T), and for every implicant I of F wrt T such that PI $\models$ I, then I $\models$ PI (PI is one of the most general implicants of F wrt T).[8])

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 $\mathsf{O}_{prop} \cup \mathcal{M}_{prop}$ is satisfiable. Q_e is a maximal conjunctive rewriting of Q iff $\mathsf{Prop}(\mathsf{Q}_e)$ is a prime implicant of $\mathsf{Prop}(\mathsf{Q}_e)$ with regard to 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 satisfiability checking algorithm for propositional theories for checking the satisfiability (also known as consistency) of PDMS schemas.

From now on, for simplicity's sake, we use the propositional notation for queries, ontologies, mappings, and rewritings. We presume that all the propositional formulas we consider are in clausal form and that PDMS satisfiability has been checked. We focus here on computing the maximal rewritings of an atomic query. (You can obtain those of a conjunctive query by combining the maximal rewritings of its atomic conjuncts.)

Our rewriting algorithm

Our "anytime" and incremental method follows an order induced by the neighborhood graph such that we obtain the maximal rewritings (and thus the answers) involving peers close to the interrogated peer first. ("Anytime" means that if you decide to stop the algorithm before the computation ends, you will obtain some answers—the ones that have been computed to that point. Usually, you have to wait until your algorithm stops to get the output.)

Proposition 1 characterizes maximal rewritings as conjunctive prime implicants of the (propositional encoding of the) query with regard to the propositional theory encoding the PDMS's ontologies and mappings. The following property, in Proposition 2, shows that you can reduce the problem of finding prime implicants to that of finding *prime implicates*. (A prime implicate PI of a formula F wrt a theory T is such that F |= PI wrt T (PI is an implicate of F wrt T), and for every implicate I of F wrt T such that I |= PI, then PI |= I (PI is one of the most specific implicates of F wrt T).[8])

Proposition 2. Conjunctive prime implicants and implicates: Let Q be a query and T a propositional theory. The conjunctive prime implicants of Q wrt T are the negation of the clausal prime implicates of $\neg Q$ wrt T.

It follows from Proposition 2 that query answering is decidable in our setting, since finding clausal prime implicates of a propositional theory is decidable.

We reuse a graph-based technique[9,10] for computing prime implicates within propositional theories partitioned into subtheories. A partitioned theory induces an *intersection graph*, in which each node represents a subtheory of the partitioning, two nodes are linked with an edge if they share propositional variables, and an edge is labeled with these shared variables.

The *forward message-passing algorithm*[9] known as MP exploits this partitioning to provide an efficient consequence-finding algorithm. MP finds logical consequences in parallel in each individual subtheory using any complete resolution strategy. MP controls the transfer of formulas between subtheories using the labels of the intersection graph's edges: it sends the logical consequences found in an individual subtheory as messages to another individual subtheory (and adds them to the set of its formulas) if they only involve propositional variables that are shared between those two subtheories. Figure 4 illustrates the algorithm's message-passing behavior on an intersection graph.

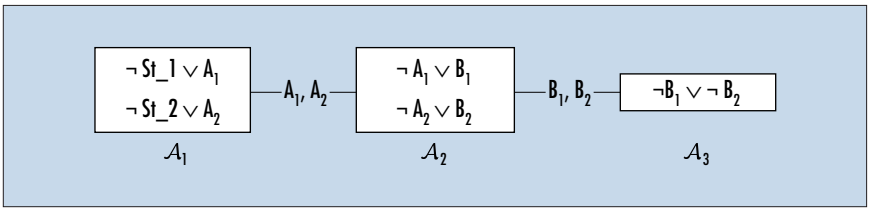


Figure 4. A partitioning and its intersection graph.

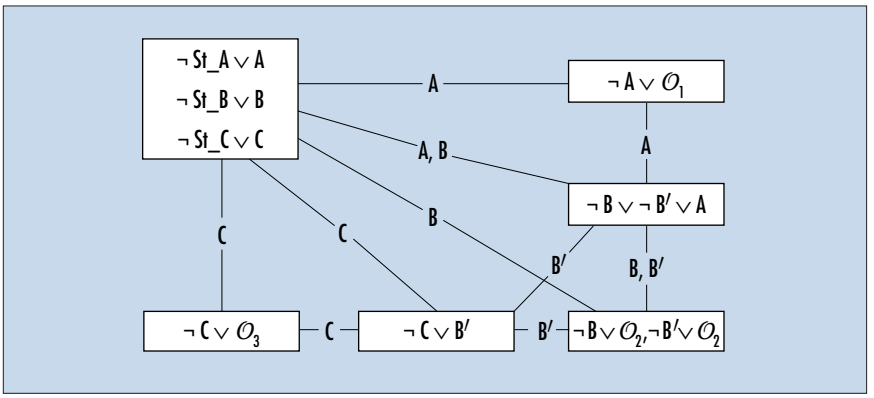


Figure 5. Intersection graph resulting from the partitioning.

Suppose you are interested in finding all the clausal prime implicates using the variables St_i—that is, the prime implicates that you can obtain in the subtheory $\mathcal{A}_1$. The algorithm starts sending messages from the most distant subtheory from $\mathcal{A}_1$: $\mathcal{A}_3$ (this means that MP starts to send messages from $\mathcal{A}_3$, which is the most distant theory from $\mathcal{A}_1$). The formula $\neg B_1 \vee \neg B_2$ is transmitted from $\mathcal{A}_3$ to $\mathcal{A}_2$. Within $\mathcal{A}_2$, the consequence-finding algorithm performed by MP infers new formulas by applying the resolution rules: $\neg A_1 \vee \neg B_2$, $\neg A_2 \vee \neg B_1$, $\neg A_1 \vee \neg A_2$. Now, only $\neg A_1 \vee \neg A_2$ is transmitted to the subtheory $\mathcal{A}_1$, and we finally obtain the prime implicate: $\neg St_1 \vee \neg St_2$.

To be complete, the MP algorithm must apply to an acyclic intersection graph. The point is that you can polynomially transform any intersection graph into a cycle-free graph such that applying MP to this acyclic graph is guaranteed to be complete. The Break-Cycles algorithm performs the appropriate transformation.[9]

Partitioning

We partition the propositional encoding of the PDMS following its natural distribution. The formulas defining an ontology's intentional classes are in a subtheory, and the mapping formulas involving the same peers are in a subtheory. (In general, a mapping might involve classes of more than two distinct ontologies.) The subtle point here is that the formulas defining the extensional classes in all ontologies are in a single subtheory called the *warehouse*.

Consider the PDMS

$\left\langle O_s^1 \cup O_s^2 \cup O_s^3, \mathcal{M}, \mathcal{E} \right\rangle$ such that

$\mathcal{O}_s^1$: $A \sqsubseteq \mathcal{O}_1$, $St\_A \sqsubseteq A$
$\mathcal{O}_s$: $B \sqsubseteq \mathcal{O}_2$, $St\_B \sqsubseteq B$, $B' \sqsubseteq \mathcal{O}_2$
$\mathcal{O}_s$: $C \sqsubseteq \mathcal{O}_3$, $St\_C \sqsubseteq C$
$\mathcal{M}$: $B \sqsubseteq B' \sqsubseteq A$, $C \sqsubseteq B'$

Figure 5 shows the intersection graph resulting from this partitioning.

Breaking cycles

We apply the following algorithm at compile time for each peer P of the PDMS.

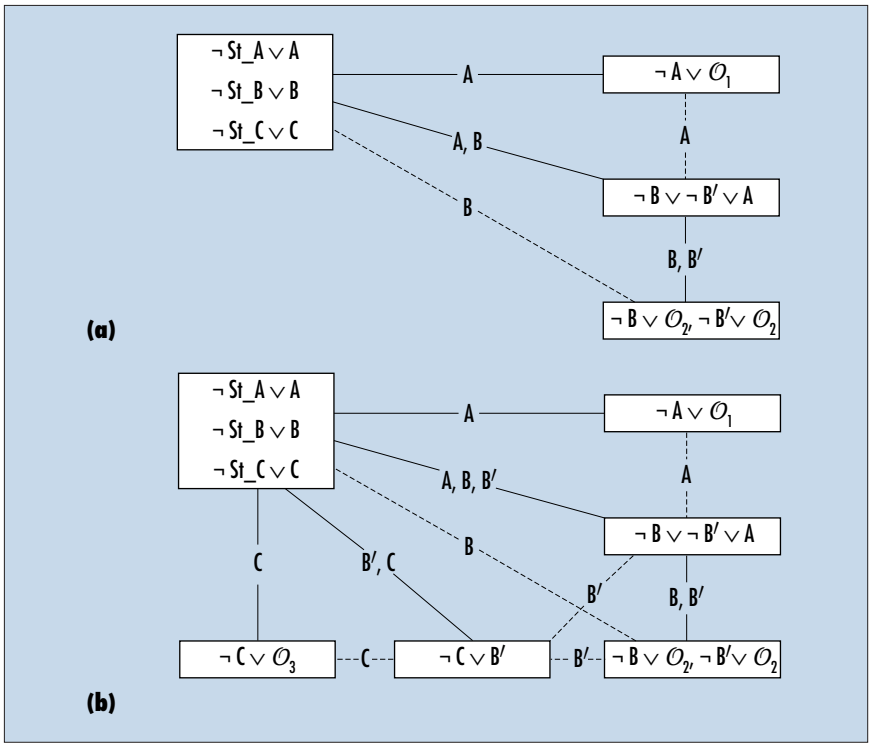


Figure 6. Cycle-free intersection graphs for O_s^1: (a) G_1 and (b) G_2.

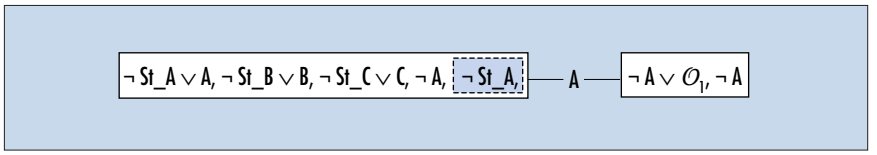


Figure 7. Local implicates (within a peer distance of 0).

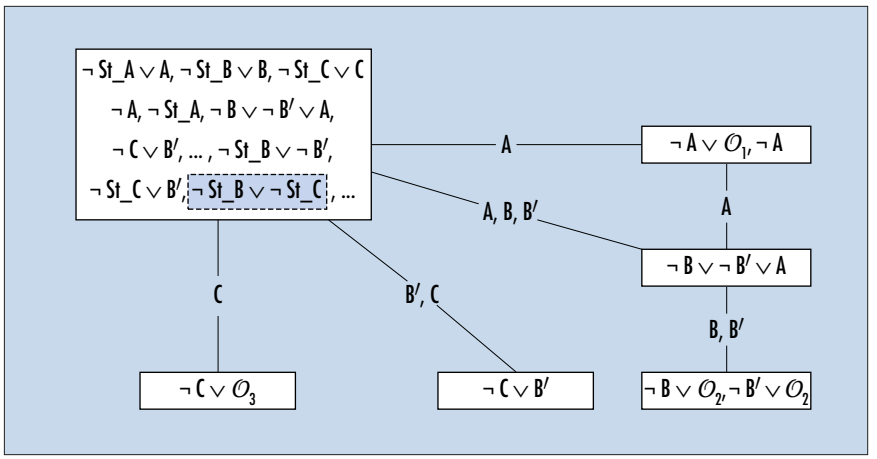


Figure 8. Implicates within a peer distance of 2.

- Let n be the distance between P and the most distant peer from P within the PDMS.
- Let G_k be the intersection graph restricted to the warehouse and the subtheories encoding the ontology of P and the ontologies (and associated mappings) of the peers whose peer distance from P is less than k.
- For $k = 0$ to n, apply Break-Cycles to G_k.

The results $\mathsf{G}_0, \ldots, \mathsf{G}_n$ computed by this algorithm are the acyclic intersection graphs to which MP will be iteratively applied at query time if P is the interrogated peer. Figures 6a and 6b show the successive results G_1 and G_2 obtained for the peer corresponding to O_s^1 in the PDMS (dotted lines indicate the removed edges for breaking cycles).

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 (we assume that the implicates of the theory alone have been computed at compile time).

Let $\mathsf{G}_0, \mathsf{G}_1, \ldots, \mathsf{G}_n$ be the successive acyclic intersection graphs computed for P at compile time. For $k = 0$ to n,

- Add $\neg\mathsf{Q}$ to the subtheory of G_k encoding the ontology of P.
- Apply the MP algorithm to G_k to compute the implicates of *distance* $\leq k$.

For example, consider that query A is posed to the peer corresponding to the ontology O_s^1 in the PDMS. Figure 7 shows the result of our algorithm at step $k = 0$.

The clause corresponding to the negation of the query, $\neg\mathsf{A}$, which has been added to the subtheory encoding O_s^1, is sent to the warehouse. A resolution is then possible in the warehouse, which produces the implicate $\neg\mathsf{St\_A}$: its negation corresponds to the only rewriting that can be produced within the interrogated peer (peer distance of 0).

Applying MP to G_1 (Figure 6a) infers no new formula.

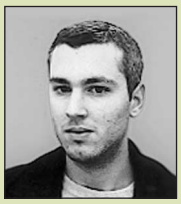
Figure 8 gives the result obtained at the last step ($k = 2$) of our algorithm. It corresponds to the application of MP to G_2 (Figure 6b): the clauses $\neg\mathsf{B} \vee \neg\mathsf{B}' \vee \mathsf{A}$ and $\neg\mathsf{C} \vee \mathsf{B}'$ are transmitted to the warehouse, thus leading to computation in the warehouse of the new implicate $\neg\mathsf{St\_B} \vee \neg\mathsf{St\_C}$ for $\neg\mathsf{A}$. Its negation is the conjunctive rewriting $\mathsf{St\_B} \wedge \mathsf{St\_C}$, whose evaluation will produce the answers obtainable within a peer distance of 2.

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 not flexible enough, and distributed systems of mediation are more appropriate. The approach presented here is an instance of the general PDMS architecture,[2,7] for which we guarantee the decidability of query answering independent of the PDMS's topology.

We plan to extend this work in two directions. First, our method imposes that each peer knows the whole PDMS schema. Such PDMSs suit applications in which the servers participating in the PDMS are well circumscribed and their ontologies are rather frozen over time (for example, e-commerce or information systems of multisite companies). However, for applications that don't meet those requirements (such as one-time file sharing between individuals), the approach does not hold. We are working on a method that lets us compute all the answers to a query in which a peer only knows the peers of the PDMS with which it shares mappings. We also plan to investigate use of the W3C metadata standard RDF (Resource Description Framework) to describe peer contents and mappings. Because RDF is part of the Semantic Web specifications, using such a standard will simplify the deployment of PDMSs dedicated to the Semantic Web.[1] ■

The Authors



François Goasdoué is an assistant professor of computer science in the Artificial Intelligence and Inference Systems group of the LRI (Comp. Sci. Lab.) at the Université Paris-Sud. He is also a member of the Gemo group (Integration of Data and Knowledge Distributed over the Web) in the Pôle Commun de Recherche en Informatique, a joint lab between INRIA (French National Institute for Research in Computer Science and Control), École Polytechnique, Université Paris-Sud, and CNRS (French National Center for Scientific Research). His research interests include knowledge representation and information integration, in particular, description logics, hybrid representation languages, query answering using views, and mediation systems. He received a PhD in computer science from the Université Paris-Sud. Contact him at LRI, Bâtiment 490, Université Paris-Sud, 91405 Orsay Cedex, France; fg@lri.fr; www.lri.fr/~goasdoue.

Marie-Christine Rousset is a professor of computer science and head of the Artificial intelligence and Inference Systems group of the LRI (Comp. Sci. Lab.) at the Université Paris-Sud. She helped initiate and coleads the Gemo group (Integration of Data and Knowledge Distributed over the Web) in the Pôle Commun de Recherche en Informatique (PCRI), a joint lab between INRIA (French National Institute for Research In Computer Science and Control), École Polytechnique, Université Paris-Sud, and CNRS (French National Center for Scientific Research). Her research topics include knowledge representation and information integration for the Semantic Web, in particular description logics, hybrid knowledge representation languages, query rewriting using views, automatic classification of semistructured data, and mediation systems for the Semantic Web. She received a PhD in computer science from the Université Paris-Sud. She received a best paper award from the AAAI in 1996. Contact her at LRI, Bâtiment 490, Université Paris-Sud, 91405 Orsay Cedex, France; mcr@lri.fr; www.lri.fr/~mcr.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 279, May 2001; www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

2. A. Halevy et al., "Schema Mediation in Peer Data Management Systems," *Proc. 19th IEEE Int'l Conf. Data Engineering* (ICDE), IEEE CS Press, 2003; www.cs.washington.edu/homes/igor/research/icde2003.pdf.

3. D. Calvanese et al., "Answering Regular Path Queries Using Views," *Proc. 16th IEEE Int'l Conf. Data Engineering* (ICDE 00), IEEE CS Press, 2000, pp. 389–398.

4. S. Abiteboul and O.M. Duschka, "Complexity of Answering Queries Using Materialized Views," *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems* (PODS 98), ACM Press, 1998, pp. 254–263.

5. J. Madhavan et al., "Representing and Reasoning about Mappings between Domain Models," *18th Nat'l Conf. Artificial Intelligence* (AAAI), AAAI Press, 2002, pp. 80–86.

6. F. Goasdoué and M.-C. Rousset, "Answering Queries Using Views: A KRDB Perspective for the Semantic Web," to be published in *ACM Trans. Internet Technology*, 2003.

7. F. Goasdoué, *Réécriture de Requêtes en Termes de Vues dans CARIN et Intégration d'Informations [Rewriting Queries Using Views in CARIN and Information Integration]*, doctoral dissertation, Université Paris-Sud XI, Orsay, 2001 (in French).

8. P. Marquis, "Knowledge Compilation Using Theory Prime Implicates," *11th Int'l Joint Conf. Artificial Intelligence* (IJCAI), Morgan Kaufmann, 1995, pp. 837–845.

9. E. Amir and S.A. McIlraith, "Partition-Based Logical Reasoning," *Proc. 7th Int'l Conf. Principles of Knowledge Representation and Reasoning* (KR 02), Morgan Kaufmann, 2000, pp. 389–400.

10. S.A. McIlraith and E. Amir, "Theorem Proving with Structured Theories," *17th Int'l Joint Conf. Artificial Intelligence* (IJCAI 01), Morgan Kaufmann, 2001, pp. 624–634.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.