SomeWhere in the Semantic Web

P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon

PCRI: Université Paris-Sud XI & CNRS (LRI), INRIA (UR Futurs)
Bâtiment 490, Université Paris-Sud XI
91405 Orsay cedex, France
{adjiman,chatalic,fg,mcr,simon}@lri.fr

Abstract. In this paper, we describe the SomeWhere semantic peer-to-peer data management system that promotes a "small is beautiful" vision of the Semantic Web based on simple personalized ontologies (e.g., taxonomies of classes) but which are distributed at a large scale. In this vision of the Semantic Web, no user imposes to others his own ontology. Logical mappings between ontologies make possible the creation of a web of people in which personalized semantic marking up of data cohabits nicely with a collaborative exchange of data. In this view, the Web is a huge peer-to-peer data management system based on simple distributed ontologies and mappings.

1 Introduction

The Semantic Web [1] envisions a world wide distributed architecture where data and computational resources will easily inter-operate based on semantic marking up of web resources using *ontologies*. Ontologies are a formalization of the semantics of application domains (e.g., tourism, biology, medecine) through the definition of classes and relations modeling the domain objects and properties that are considered as meaningful for the application. Most of the concepts, tools and techniques deployed so far by the Semantic Web community correspond to the "big is beautiful" idea that high expressivity is needed for describing domain ontologies. As a result, when they are applied, the current Semantic Web technologies are mostly used for building thematic portals but do not scale up to the web. In contrast, SomeWhere promotes a "small is beautiful" vision of the Semantic Web [2] based on simple personalized ontologies (e.g., taxonomies of atomic classes) but which are distributed at a large scale. In this vision of the Semantic Web introduced in [3], no user imposes to others his own ontology but logical mappings between ontologies make possible the creation of a web of people in which personalized semantic marking up of data cohabits nicely with a collaborative exchange of data. In this view, the web is a huge peer-to-peer data management system based on simple distributed ontologies and mappings.

Peer-to-peer data management systems have been proposed recently [4–7] to generalize the centralized approach of information integration systems based on single mediators. In a peer-to-peer data management system, there is no central mediator: each peer has its own ontology and data or services, and can mediate

with some other peers to ask and answer queries. The existing systems vary according to (a) the expressive power of their underlying data model and (b) the way the different peers are semantically connected. Both characteristics have impact on the allowed queries and their distributed processing.

In Edutella [8], each peer stores locally data (educational resources) that are described in RDF relatively to some reference ontologies (e.g., http://dmoz.org). For instance, a peer can declare that it has data related to the concept of the dmoz taxonomy corresponding to the path *Computers/Programming/ Languages/Java*, and that for such data it can export the *author* and the *date* properties. The overlay network underlying Edutella is a hypercube of super-peers to which peers are directly connected. Each super-peer is a mediator over the data of the peers connected to it. When it is queried, its first task is to check if the query matches with its schema: if that is the case, it transmits the query to the peers connected to it, which are likely to store the data answering the query ; otherwise, it routes the query to some of its neighbour super-peers according to a strategy exploiting the hypercube topology for guaranteeing a worst-case logarithmic time for reaching the relevant super-peer.

In contrast with Edutella, Piazza [4, 9] does not consider that the data distributed over the different peers must be described relatively to some existing reference schemas. Each peer has its own data and schema and can mediate with some other peers by declaring *mappings* between its schema and the schemas of those peers. The topology of the network is not fixed (as in Edutella) but accounts for the existence of mappings between peers: two peers are logically connected if there exists a mapping between their two schemas. The underlying data model of the first version of Piazza [4] is relational and the mappings between relational peer schemas are inclusion or equivalence statements between conjunctive queries. Such a mapping formalism encompasses the *Local-as-View* and the *Global-as-View* [10] formalisms used in information integration systems based on single mediators. The price to pay is that query answering is undecidable except if some restrictions are imposed on the mappings or on the topology of the network [4]. The currently implemented version of Piazza [9] relies on a tree-based data model: the data is in XML and the mappings are equivalence and inclusion statements between XML queries. Query answering is implemented based on practical (but not complete) algorithms for XML query containment and rewriting. The scalability of Piazza so far does not go up to more than about 80 peers in the published experiments and relies on a wide range of optimizations (mappings composition [11], paths pruning [12]), made possible by the centralized storage of all the schemas and mappings in a global server.

In SomeWhere, we have made the choice of being fully distributed: there are neither super-peers nor a central server having the global view of the overlay network. In addition, we aim at scaling up to thousands of peers. To make it possible, we have chosen a simple class-based data model in which the data is a set of resource identifiers (e.g., URIs), the schemas are (simple) definitions of classes possibly constrained by inclusion, disjunction or equivalence statements, and mappings are inclusion, disjunction or equivalence statements between classes

of different peer schemas. That data model is in accordance with the W3C recommendations since it is captured by the propositional fragment of the OWL ontology language (http://www.w3.org/TR/owl-semantics).

The paper is organized as follows. Section 2 defines the SomeWhere data model. In Section 3, we show how the corresponding query rewriting problem can be reduced by a propositional encoding to distributed reasoning in propositional logic. In Section 4, we describe the properties of the message based distributed reasoning algorithm that is implemented in SomeWhere, and we report experiments on networks of 1000 peers. Section 5 surveys some recent related work on peer-to-peer data management systems. We conclude and present our forthcoming work in Section 6.

2 SomeWhere Data model

In SomeWhere a new peer joins the network through some peers that it knows (its acquaintances) by declaring mappings between its own ontology and the ontologies of its acquaintances. Queries are posed to a given peer using its local ontology. The answers that are expected are not only instances of local classes but possibly instances of classes of peers distant from the queried peer if it can be infered from the peer ontologies and the mappings that those instances are answers of the query. Local ontologies, storage descriptions and mappings are defined using a fragment of OWL DL which is the description logic fragment of the Ontology Web Language recommended by W3C. We call OWL PL the fragment of OWL DL that we consider in SomeWhere, where PL stands for propositional logic. OWL PL is the fragment of OWL DL reduced to the disjunction, conjunction and negation constructors for building class descriptions.

2.1 Peer ontologies

Each peer ontology is made of a set of class definitions and possibly a set of equivalence, inclusion or disjointness axioms between class descriptions. A class description is either the universal class ($\top$), the empty class ($\bot$), an atomic class or the union ($\sqcup$), intersection ($\sqcap$) or complement ($\neg$) of class descriptions.

The name of atomic classes are unique to each peer: we use the notation $P{:}A$ for identifying an atomic class A of the ontology of a peer P. The *vocabulary* of a peer P is the set of names of its atomic classes.

Class descriptions

	Logical notation	*OWL notation*
universal class	$\top$	$Thing$
empty class	$\bot$	$Nothing$
atomic class	$P{:}A$	$classID$
conjunction	$D1 \sqcap D2$	$intersectionOf(D1\ D2)$
disjunction	$D1 \sqcup D2$	$unionOf(D1\ D2)$
negation	$\neg D$	$complementOf(D)$

Axioms of class definitions

	Logical notation	*OWL notation*
Complete	$P{:}A \equiv D$	$Class(P{:}A\ complete\ D)$
Partial	$P{:}A \sqsubseteq D$	$Class(P{:}A\ partial\ D)$

Axioms on class descriptions

	Logical notation	*OWL notation*
equivalence	$D1 \equiv D2$	$EquivalentClasses(D1\ D2)$
inclusion	$D1 \sqsubseteq D2$	$SubClassOf(D1\ D2)$
disjointness	$D1 \sqcap D2 \equiv \bot$	$DisjointClasses(D1\ D2)$

2.2 Peer storage descriptions

The specification of the data that is stored locally in a peer P is done through the declaration of atomic *extensional classes* defined in terms of atomic classes of the peer ontology, and assertional statements relating data identifiers (e.g., URIs) to those extensional classes. We restrict the axioms defining the extensional classes to be inclusion statements between an atomic extensional class and a description combining atomic classes of the ontology. We impose that restriction in order to fit with a *Local-as-View* approach and an open-world assumption within the information integration setting [10]. We will use the notation $P{:}ViewA$ to denote an extensional class $ViewA$ of the peer P.

Storage description

declaration of extensional classes:

Logical notation	*OWL notation*
$P{:}ViewA \sqsubseteq C$	$SubClassOf(P{:}ViewA\ \ C)$

assertional statements:

Logical notation	*OWL notation*
$P{:}ViewA(a)$	$individual(a\ type(P{:}ViewA))$

2.3 Mappings

Mappings are disjointness, equivalence or inclusion statements involving atomic classes of different peers. They express the semantic correspondence that may exist between the ontologies of different peers.

The *acquaintance graph* accounts for the connection induced by the mappings between the different peers within a given SomeWhere peer-to-peer network.

Definition 1 (Acquaintance graph). *Let $\mathcal{P} = \{P_i\}_{i\in[1..n]}$ a collection of peers with their respective vocabularies Voc_{P_i}. Let $Voc = \bigcup_{i=1}^{n} Voc_{P_i}$ be the vocabulary of $\mathcal{P}$. Its* acquaintance graph *is a graph $\Gamma = (\mathcal{P}, \text{ACQ})$ where $\mathcal{P}$ is the set of vertices and $\text{ACQ} \subseteq Voc \times \mathcal{P} \times \mathcal{P}$ is a set of labelled edges such that for every $(c, P_i, P_j) \in \text{ACQ}$, $i \neq j$ and $c \in Voc_{P_i} \cap Voc_{P_j}$.*

A labelled edge (c, P_i, P_j) expresses that peers P_i and P_j know each other to be sharing the class c. This means that c belongs to the intentional classes of P_i (or P_j) and is involved in a mapping with intentional classes of P_j (or P_i).

2.4 Schema of a SomeWhere network

In a SomeWhere network, the schema is not centralized but distributed through the union of the different peer ontologies and the mappings. The important point is that each peer has a partial knowledge of the schema: it just knows its own local ontology and the mappings with its acquaintances.

Let $\mathcal{P}$ be a SomeWhere peer-to-peer network made of a collection of peers $\{P_i\}_{i\in[1..n]}$. For each peer P_i, let O_i, V_i and M_i be the sets of axioms defining respectively the local ontology of P_i, the declaration of its extensional classes and the set of mappings stated at P_i between classes of O_i and classes of the ontologies of the acquaintances of P_i. The schema $\mathcal{S}$ of $\mathcal{P}$ is the union $\bigcup_{i\in[1..n]} O_i \cup V_i \cup M_i$ of the ontologies, the declaration on extensional classes and of the sets of mappings of all the peers of $\mathcal{P}$.

2.5 Semantics

The semantics is a standard logical formal semantics defined in terms of *interpretations*. An interpretation I is a pair $(\Delta^I, .^I)$ where Δ is a non-empty set, called the domain of interpretation, and $.^I$ is an interpretation function which assigns a subset of Δ^I to every class identifier and an element of Δ^I to every data identifier.

An interpretation I is a *model* of the distributed schema of a SomeWhere peer-to-peer network $\mathcal{P} = \{P_i\}_{i\in[1..n]}$ iff each axiom in $\bigcup_{i\in[1..n]} O_i \cup V_i \cup M_i$ is satisfied by I.

Interpretations of axioms rely on interpretations of class descriptions which are inductively defined as follows:

- $\top^I = \Delta^I$, $\bot^I = \emptyset$
- $(\neg C)^I = \Delta^I \backslash C^I$
- $(C_1 \sqcup C_2)^I = C_1^I \cup C_2^I$, $(C_1 \sqcap C_2)^I = C_1^I \cap C_2^I$

Axioms are satisfied if the following holds:

- $C \sqsubseteq D$ is satisfied in I iff $C^I \subseteq D^I$
- $C \equiv D$ is satisfied in I iff $C^I = D^I$
- $C \sqcap D \equiv \bot$ is satisfied in I iff $C^I \cap D^I = \emptyset$

A SomeWhere peer-to-peer network is *satisfiable* iff its schema has a model.

Given a SomeWhere peer-to-peer network $\mathcal{P} = \{P_i\}_{i\in[1..n]}$, a class description C *subsumes* a class description D iff in each model I of the schema of $\mathcal{P}$, $D^I \subseteq C^I$.

2.6 Illustrative example

We illustrate the SomeWhere data model on a small example of four peers modeling four persons Ann, Bob, Chris and Dora, each of them bookmarking URLs about restaurants they know or like, according to their own taxonomy for categorizing restaurants.

Ann, who is working as a restaurant critics, organizes its restaurant URLs according to the following classes:

- the class $Ann{:}G$ of restaurants considered as offering a "good" cooking, among which she distinguishes the subclass $Ann{:}R$ of those which are rated: $Ann{:}R \sqsubseteq Ann{:}G$

• the class $Ann{:}R$ is the union of three disjoint classes $Ann{:}S1$, $Ann{:}S2$, $Ann{:}S3$ corresponding respectively to the restaurants rated with $1, 2$ or 3 stars:

$Ann{:}R \equiv Ann{:}S1 \sqcup Ann{:}S2 \sqcup Ann{:}S3$

$Ann{:}S1 \sqcap Ann{:}S2 \equiv \bot \quad Ann{:}S1 \sqcap Ann{:}S3 \equiv \bot$

$Ann{:}S2 \sqcap Ann{:}S3 \equiv \bot$

• the classes $Ann{:}I$ and $Ann{:}O$, respectively corresponding to Indian and Oriental restaurants

• the classes $Ann{:}C$, $Ann{:}T$ and $Ann{:}V$ which are subclasses of $Ann{:}O$ denoting Chinese, Taï and Vietnamese restaurants respectively: $Ann{:}C \sqsubseteq Ann{:}O$, $Ann{:}T \sqsubseteq Ann{:}O$, $Ann{:}V \sqsubseteq Ann{:}O$

Suppose that the data stored by Ann that she accepts to make available deals with restaurants of various specialties, and only with those rated with 2 stars among the rated restaurants. The extensional classes declared by Ann are then:
$Ann{:}ViewS2 \sqsubseteq Ann{:}S2$, $Ann{:}ViewC \sqsubseteq Ann{:}C$,
$Ann{:}ViewV \sqsubseteq Ann{:}V$, $Ann{:}ViewT \sqsubseteq Ann{:}T$,
$Ann{:}ViewI \sqsubseteq Ann{:}I$

Bob, who is found of Asian cooking and likes high quality, organizes his restaurant URLs according to the following classes:

• the class $Bob{:}A$ of Asian restaurants

• the class $Bob{:}Q$ of high quality restaurants that he knows

Suppose that he wants to make available every data that he has stored. The extensional classes that he declares are $Bob{:}ViewA$ and $Bob{:}ViewQ$ (as subclasses of $Bob{:}A$ and $Bob{:}Q$): $Bob{:}ViewA \sqsubseteq Bob{:}A$, $Bob{:}ViewQ \sqsubseteq Bob{:}Q$

Chris is more found of fish restaurants but recently discovered some places serving a very nice cantonese cuisine. He organizes its data with respect to the following classes:

• the class $Chris{:}F$ of fish restaurants,

• the class $Chris{:}CA$ of Cantonese restaurants

Suppose that he declares the extensional classes $Chris{:}ViewF$ and $Chris{:}ViewCA$ as subclasses of $Chris{:}F$ and $Chris{:}CA$ respectively: $Chris{:}ViewF \sqsubseteq Chris{:}F$, $Chris{:}ViewCA \sqsubseteq Chris{:}CA$

Dora organizes her restaurants URLs around the class $Dora{:}DP$ of her preferred restaurants, among which she distinguishes the subclass $Dora{:}P$ of pizzerias and the subclass $Dora{:}SF$ of seafood restaurants.

Suppose that the only URLs that she stores concerns pizzerias: the only extensional class that she has to declare is $Dora{:}ViewP$ as a subclass of $Dora{:}P$: $Dora{:}ViewP \sqsubseteq Dora{:}P$

Ann, **Bob**, **Chris** and **Dora** express what they know about each other using mappings stating properties of class inclusion or equivalence.

Ann is very confident in Bob's taste and agrees to include Bob' selection as good restaurants by stating $Bob{:}Q \sqsubseteq Ann{:}G$. Finally, she thinks that Bob's Asian restaurants encompass her Oriental restaurant concept: $Ann{:}O \sqsubseteq Bob{:}A$

Bob knows that what he calls Asian cooking corresponds exactly to what Ann classifies as Oriental cooking. This may be expressed using the equivalence

statement : $Bob{:}A \equiv Ann{:}O$ (note the difference of perception of Bob and Ann regarding the mappings between $Bob{:}A$ and $Ann{:}O$)

Chris considers that what he calls fish specialties is a particular case of Dora seafood specialties: $Chris{:}F \sqsubseteq Dora{:}SF$

Dora counts on both Ann and Bob to obtain good Asian restaurants : $Bob{:}A \sqcap Ann{:}G \sqsubseteq Dora{:}DP$

Figure 1 describes the resulting acquaintance graph. In order to alleviate the notations, we omit the local peer name prefix except for the mappings. Edges are labeled with the class identifiers that are shared through the mappings.

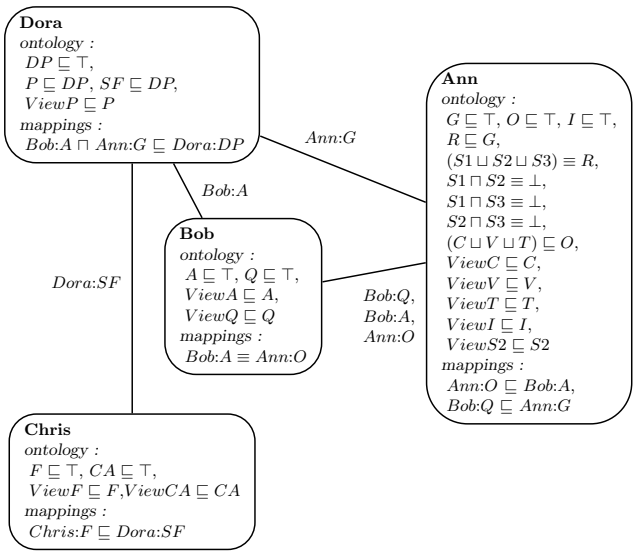

Fig. 1. The restaurants network

3 Query rewriting

In SomeWhere, each user interrogates the peer-to-peer network through one peer of his choice, and uses the vocabulary of this peer to express his query. Therefore, queries are logical combinations of classes of a given peer ontology.

The corresponding answer sets are expressed in intention in terms of the combinations of extensional classes that are *rewritings* of the query. The point is that extensional classes of several distant peers can participate to the rewritings, and thus to the answer of a query posed to a given peer.

Given a SomeWhere peer-to-peer network $\mathcal{P} = \{P_i\}_{i \in [1..n]}$, a logical combination Q_e of extensional classes is a *rewriting* of a query Q iff Q subsumes Q_e. Q_e is a *maximal rewriting* if there does not exist another rewriting Q'_e of Q (strictly) subsuming Q_e.

In the SomeWhere setting, query rewriting can be equivalently reduced to distributed reasoning over logical propositional theories by a straighforward propositional encoding of the distributed schema of a SomeWhere network.

Before presenting the propositional encoding in Section 3.2 and the distributed consequence finding algorithm in Section 4, we illustrate the corresponding query processing on the example of Section 2.6.

3.1 Illustrative example (continued)

Consider that a user queries the restaurants network through the **Dora** peer by asking the query $Dora{:}DP$, meaning that he is interested in getting as answers the set of favourite restaurants of Dora:

• Using $Dora{:}P \sqsubseteq Dora{:}DP$ and $Dora{:}ViewP \sqsubseteq Dora{:}P$, we obtain $Dora{:}ViewP$ as a local rewriting corresponding to the extensional class of pizzeria URLs stored by Dora.

• Using $Dora{:}SF \sqsubseteq Dora{:}DP$, the fact that $Dora{:}SF$ is shared with $Chris$ by the mapping $Chris{:}F \sqsubseteq Dora{:}SF$, and $Chris{:}ViewF \sqsubseteq Chris{:}F$, we obtain $Chris{:}ViewF$ as a new rewriting meaning that another way to get restaurants liked by Dora is to obtain the Fish restaurants stored by Chris.

• Finally, using the mapping $Bob{:}A \sqcap Ann{:}G \sqsubseteq Dora{:}DP$, the query leads to look for rewritings of $Bob{:}A \sqcap Ann{:}G$, where both $Bob{:}A$ and $Ann{:}G$ are shared with neighbor peers. In such cases our algorithm uses a split/recombination approach. Each shared component (here $Bob{:}A$ and $Ann{:}G$) is then processed independly as a subquery, transmitted to its appropriate neighbors and associated with some queue data structure, where its returned rewritings are accumulated. As soon as at least one rewriting has been obtained for each component, the respective queued rewritings of each component are recombined to produce rewritings of the initial query. This recombination process continues incrementally, as new rewritings for a component are produced. Note that since each subcomponent is processed asynchronously, the order in which recombined rewritings are produced is unpredictable. For the sake of simplicity, in the following we consider sequentially the results obtained for the two subqueries $Bob{:}A$ and $Ann{:}G$:

– On the Bob peer, because of $Bob{:}ViewA \sqsubseteq Bob{:}A$, $Bob{:}ViewA$ is a local rewriting of $Bob{:}A$, which is transmitted back to the Dora peer, where it is queued for a future combination with rewritings of the other subquery $Ann{:}G$.

In addition, guided by the mapping $Ann{:}O \equiv Bob{:}A$, the Bob peer transmits to the Ann peer the query $Ann{:}O$. The Ann peer processes that query locally and transmits back to the Bob peer the rewriting $Ann{:}ViewC \sqcup Ann{:}ViewT \sqcup Ann{:}ViewV$, which in turn is transmitted back to the Dora peer as an additional rewriting for the subquery $Bob{:}A$ and queued there.

– On the Ann peer, using $Ann{:}R \sqsubseteq Ann{:}G$, $(Ann{:}S1 \sqcup Ann{:}S2 \sqcup Ann{:}S3) \equiv Ann{:}R$ and $Ann{:}ViewS2 \sqsubseteq Ann{:}S2$, $Ann{:}ViewS2$ is obtained as a local rewriting of $Ann{:}G$. It is transmitted back to the Dora peer where it is queued for recombination. Let us suppose that the two rewritings of $Bob{:}A$ ($Bob{:}ViewA$ and $Ann{:}ViewC \sqcup Ann{:}ViewT \sqcup Ann{:}ViewV$) have aleady been produced at that time. Their combination with $Ann{:}ViewS2$ gives two rewritings which are sent back to the user:

∗ $Ann{:}ViewS2 \sqcap Bob{:}ViewA$, meaning that a way to obtain restaurants liked by Dora is to find restautants that are both stored by Ann as rated with 2 stars and by Bob as Asian restaurants,

$*$ $Ann{:}ViewS2 \sqcap (Ann{:}ViewC \sqcup Ann{:}ViewT \sqcup Ann{:}ViewV)$ meaning that another way to obtain restaurants liked by Dora is to find restautants stored by Ann as restaurants rated with 2 stars and also as Chinese, Thai or Vietnamese restaurants. Note that this rewriting, although obtained via different peers after splitting/recombination, turns out to be composed only of extensional classes of the same peer: Ann.

Still on the Ann peer, because of the mapping $Bob{:}Q \sqsubseteq Ann{:}G$, Ann transmits the query $Bob{:}Q$ to Bob, which transmits back to Ann $Bob{:}ViewQ$ as a rewriting of $Bob{:}Q$ (and thus of $Ann{:}G$). Ann then transmits $Bob{:}ViewQ$ back to Dora as a rewriting of $Ann{:}G$, where it is queued for combination. On Dora's side, $Bob{:}ViewQ$ is now combined with the queued rewritings of $Bob{:}A$ ($Bob{:}ViewA$ and $Ann{:}ViewC \sqcup Ann{:}ViewT \sqcup Ann{:}ViewV$). As a result, two new rewritings are sent back to the user:

$*$ $Bob{:}ViewQ \sqcap Bob{:}ViewA$ meaning that to obtain restaurants liked by Dora one can take the restaurants that Bob stores as high quality restaurants and as Asian restaurants,

$*$ $Bob{:}ViewQ \sqcap (Ann{:}ViewC \sqcup Ann{:}ViewT \sqcup Ann{:}ViewV)$ providing a new way of getting restaurants liked by Dora: those that are both stored as high quality restaurants by Bob and as Chinese, Thai or Vietnamese restaurants by Ann.

3.2 Propositional encoding of query rewriting in SomeWhere

The propositional encoding concerns the schema of a SomeWhere network and the queries. It consists in transforming each query and schema statement into a propositional formula using class identifiers as propositional variables.

The propositional encoding of a class description D, and thus of a query, is the propositional formula $Prop(D)$ obtained inductively as follows:

- $Prop(\top) = true$, $Prop(\bot) = false$
- $Prop(A) = A$, if A is an atomic class
- $Prop(D_1 \sqcap D_2) = Prop(D_1) \wedge Prop(D_2)$
- $Prop(D_1 \sqcup D_2) = Prop(D_1) \vee Prop(D_2)$
- $Prop(\neg D) = \neg(Prop(D))$

The propositional encoding of the schema $\mathcal{S}$ of a SomeWhere peer-to-peer network $\mathcal{P}$ is the distributed propositional theory $Prop(\mathcal{S})$ made of the formulas obtained inductively from the axioms in $\mathcal{S}$ as follows:

- $Prop(C \sqsubseteq D) = Prop(C) \Rightarrow Prop(D)$
- $Prop(C \equiv D) = Prop(C) \Leftrightarrow Prop(D)$
- $Prop(C \sqcap D \equiv \bot) = \neg Prop(C) \vee \neg Prop(D)$

From now on, for simplicity purpose, we use the propositional clausal form notation for the queries and SomeWhere peer-to-peer network schemas.

As an illustration, let us consider the propositional encoding of the example presented in Section 2.6. Once in clausal form and after the removal of tautologies, we obtain (Figure 2) the acquaintance graph where each peer schema is described as a propositional theory.

Proposition 1 states that the propositional encoding transfers satisfiability and establishes the connection between (maximal) conjunctive rewritings and clausal proper (prime) implicates.

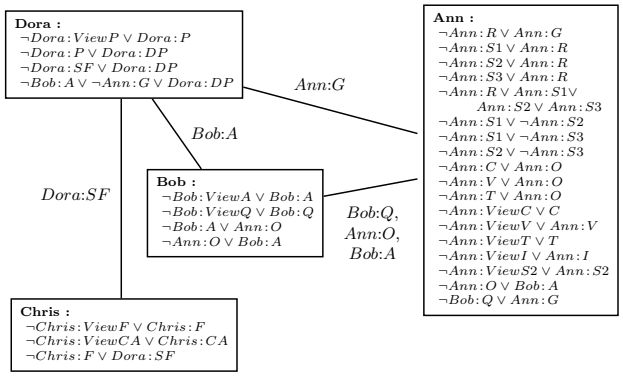

Fig. 2. Propositional encoding for the restaurant network

Definition 2 (Proper prime implicate wrt a theory). *Let T be a clausal theory and q be a clause. A clause m is said to be:*

- *a* prime implicate *of q wrt T iff $T \cup \{q\} \models m$ and for any other clause m', if $T \cup \{q\} \models m'$ and $m' \models m$ then $m' \equiv m$.*
- *a* proper prime implicate *of q wrt T iff it is a prime implicate of q wrt T and $T \not\models m$.*

Proposition 1 (Propositional transfer). *Let $\mathcal{P}$ be a SomeWhere peer-to-peer network and let $Prop(S(\mathcal{P}))$ be the propositional encoding of its schema. Let V_e be the set of all the extensional classes.*

- *$S(\mathcal{P})$ is satisfiable iff $Prop(S(\mathcal{P}))$ is satisfiable.*
- *q_e is a maximal conjunctive rewriting of a query q iff $\neg Prop(q_e)$ is a* proper prime implicate of $\neg Prop(q)$ wrt $Prop(S(\mathcal{P}))$ *such that all its variables are extensional classes.*

Proposition 1 gives us a way to compute *all* the answers of a query. The maximal conjunctive rewritings of a query q within a peer-to-peer network $\mathcal{P}$ correspond to the negation of the proper prime implicates of $\neg q$ wrt the propositional encoding of the schema of $S(\mathcal{P})$. Since the number of proper prime implicates of a clause wrt a clausal theory is finite, every query in SomeWhere has a finite number of maximal conjunctive rewritings. Therefore, according to [13], the set of *all* of its answers is exactly the union of the answer sets of its rewritings and is obtained in PTIME data complexity.

In the following section, we present a distributed consequence finding algorithm which computes the set of proper prime implicates of a literal wrt a distributed propositional clausal theory. According to Proposition 1, if this algorithm is applied to a distributed theory resulting from the propositional encoding of the schema of a SomeWhere network, with the extensional classes symbols as *target variables*, and triggered with a literal $\neg q$, it computes in fact the negation of the maximal conjunctive rewritings of the *atomic* query q. Since in our setting the maximal rewritings of an arbitrary query can be obtained by combining the maximal rewritings of its atomic components, we focus on the computation of the rewritings of atomic queries.

4 Algorithmic machinery and experiments

The SomeWhere peer-to-peer data management system relies on a distributed algorithm presented in [14]. For this paper to be self-contained, we describe the three message passing procedures of the algorithm which are implemented locally at each peer. They are triggered by the reception of a *query* (resp. *answer*, *final*) message, sent by a Sender peer to a receiver peer, denoted by Self, which executes the procedure. Procedures handle an history initialized to the empty sequence. An history *hist* is a sequence of triples (l, P, c) (where l is a literal, P a peer, and c a clause). An history $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1), (l_0, P_0, c_0)]$ represents a branch of reasoning initiated by the propagation of the literal l_0 within the peer P_0, and the splitting of the clause c_0: for every $i \in [0..n-1]$, c_i is a consequence of l_i and P_i, and l_{i+1} is a literal of c_i, which is propagated in P_{i+1}.
ReceiveQueryMessage is triggered by the reception of a *query* message $m(Sender, Receiver, query, hist, l)$ sent by the peer $Sender$ to the peer $Receiver$ which executes the procedure: on the demand of $Sender$, with which it shares the variable of l, it processes the literal l.
ReceiveAnswerMessage is triggered by the reception of an *answer* message $m(Sender, Receiver, answer, hist, r)$ sent by the peer $Sender$ to the peer $Receiver$ which executes the procedure: it processes the answer r (which is a clause the variables of which are target variables) sent back by $Sender$ for the literal l (last added in the history) ; it may have to combine it with other answers for literals being in the same clause as l.
ReceiveFinalMessage is triggered by the reception of a $final$ message $m(Sender, Receiver, final, hist, true)$: the peer $Sender$ notifies the peer $Receiver$ that computation for the literal l (last added in the history) is completed.

Those procedures handle two local data structures:
answer$(l, hist)$ caches answers resulting from the propagation of l within the reasoning branch corresponding to $hist$;
final$(q, hist)$ is set to true when the propagation of q within the reasoning branch of the history $hist$ is completed. The reasoning is initiated by the user (denoted by a particular peer $User$) sending to a given peer P a message $m(User, P, query, \emptyset, q)$, which triggers the procedure ReceiveQueryMessage($m(User, P, query, \emptyset, q)$) that is locally executed by P.

In the following procedures, since they are locally executed by the peer which receives the message, we denote by $Self$ the receiver peer. We also assume that:

- for a literal q, $Resolvent(q, P)$ denotes the set of clauses obtained by resolution between q and a clause of P,
- for a literal q, $\bar{q}$ denotes its complementary literal,
- for a clause c of a peer P, $S(c)$ (resp. $L(c)$) denotes the disjonction of literals of c whose variables are shared (resp. not shared) with any acquaintance of P. $S(c) = \square$ thus expresses that c does not contain any shared variable,
- $\mathcal{T}arget(P)$ is the language of clauses (including $\square$) involving only variables that are extensonial classes of P.
- $\otimes$ is the distribution operator on sets of clauses: $S_1 \otimes \cdots \otimes S_n = \{c_1 \vee \cdots \vee c_n \mid c_1 \in S_1, \ldots, c_n \in S_n\}$. If $L = \{l_1, \ldots, l_p\}$, $\otimes_{l \in L} S_l$ denotes $S_{l_1} \otimes \cdots \otimes S_{l_p}$.

Algorithm 1: Message passing procedure for processing queries
ReceiveQueryMessage($m(Sender, Self, query, hist, q)$)
(1) **if** $(\bar{q}, \_, \_) \in hist$
(2) **send** $m(Self, Sender, answer, [(q, Self, \Box)|hist], \Box)$
(3) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(4)**else if** $q \in Self$ or $(q, Self, \_) \in hist$
(5) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(6)**else**
(7) local$(Self) \leftarrow \{q\} \cup Resolvent(q, Self)$
(8) **if** $\Box \in$ local$(Self)$
(9) **send** $m(Self, Sender, answer, [(q, Self, \Box)|hist], \Box)$
(10) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(11) **else**
(12) local$(Self) \leftarrow \{c \in$ local$(Self)|\ L(c) \in \mathcal{T}arget(Self)\}$
(13) **if** for every $c \in$ local$(Self), S(c) = \Box$
(14) **foreach** $c \in$ local$(Self)$
(15) **send** $m(Self, Sender, answer, [(q, Self, c)|hist], c)$
(16) **send** $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(17) **else**
(18) **foreach** $c \in$ local$(Self)$
(19) **if** $S(c) = \Box$
(20) **send** $m(Self, Sender, answer, [(q, Self, c)|hist], c)$
(21) **else**
(22) **foreach** literal $l \in S(c)$
(23) **if** $l \in \mathcal{T}arget(Self)$
(24) answer$(l, [(q, Self, c)|hist]) \leftarrow \{l\}$
(25) **else**
(26) answer$(l, [(q, Self, c)|hist]) \leftarrow \emptyset$
(27) final$(l, [(q, Self, c)|hist]) \leftarrow false$
(28) **foreach** $RP \in$ acq$(l, Self)$
(29) **send** $m(Self, RP, query, [(q, Self, c)|hist], l)$

Algorithm 2: Message passing procedure for processing answers
ReceiveAnswerMessage($m(Sender, Self, answer, hist, r)$)
(1)$hist$ is of the form $[(l', Sender, c'), (q, Self, c)|hist']$
(2)answer$(l', hist) \leftarrow$ answer $(l', hist) \cup \{r\}$
(3)result$\leftarrow \Circledcirc_{l \in S(c) \setminus \{l'\}}$answer$(l, hist) \Circledcirc \{L(c) \vee r\}$
(4)**if** $hist' = \emptyset$, $U \leftarrow User$ **else** $U \leftarrow$ the first peer P' of $hist'$
(5)**foreach** $cs \in$ result
(6) **send** $m(Self, U, answer, [(q, Self, c)|hist'], cs)$

Algorithm 3: Message passing procedure for notifying termination
ReceiveFinalMessage($m(Sender, Self, final, hist, true)$)
(1)$hist$ is of the form $[(l', Sender, true), (q, Self, c)|hist']$
(2)final$(l', hist) \leftarrow true$
(3)**if** for every $l \in S(c)$, final$(l, hist) = true$
(4) **if** $hist' = \emptyset$ $U \leftarrow User$ **else** $U \leftarrow$ the first peer P' of $hist'$
(5) **send** $m(Self, U, final, [(q, Self, true)|hist'], true)$
(6) **foreach** $l \in S(c)$
(7) answer$(l, [(l, Sender, _), (q, Self, c)|hist']) \leftarrow \emptyset$

The following theorems summarize the main properties of this distributed message passing algorithm and thus of the SomeWhere peer-to-peer data management system. Theorem 1 states the termination and the soundness of the algorithm. Theorem 2 states its completeness under the condition that each peer theory is saturated by resolution. Theorem 3 states that the user is notified of the termination when it occurs, which is crucial for an anytime algorithm. Full proofs are given in [15]. In the following theorems, let $\mathcal{T}$ be the propositional encoding of the schema $S(\mathcal{P})$ of a peer-to-peer SomeWhere network, let $\neg q$ the negation of an atomic query q, let T be the propositional encoding of the local schema and mappings of the asked peer.

Theorem 1 (Soundness). *If T receives from the user the message $m(User, T, query, \emptyset, \neg q)$, then:*

- *a finite number of answer messages will be produced ;*
- *each produced answer message $m(T, User, answer, [(\neg q,\ T, \_)], r)$ is such that r is an implicate of $\neg q$ wrt $S(\mathcal{P})$ which belong to $\mathcal{T}arget(\mathcal{P})$.*

Theorem 2 (Completeness). *If each local theory is saturated by resolution and if T receives from the user the message $m(User, T, query, \emptyset, \neg q)$, then for each proper prime implicate r of $\neg q$ wrt $S(\mathcal{P})$ belonging to $\mathcal{T}arget(\mathcal{P})$, an answer message $m(T, User, answer, [(\neg q, T, \_)], r)$ will be produced.*

Theorem 3 (Termination notification). *If r is the last result returned in an answer message $m(T, User, answer,\ [(\neg q, T, \_)], r)$ then the user will be notified of the termination by a message $m(T, User, final, [(\neg q, T, true)], true)$.*

It is important to notice that $\square$ can be returned by our algorithm as a proper prime implicate because of the lines (1) to (3) and (8) to (10) in ReceiveQueryMessage. In that case, as a corollary of the above theorems, the union the propositional encoding of the schema of the SomeWhere network and the query is detected unsatisfiable. Therefore, our algorithm can be exploited for checking the satisfiability of the global schema at each join of a new peer.

5 Related work

As we have pointed it out in the introduction, the SomeWhere peer data management system distinguishes from Edutella [8] by the fact that there is no need of super-peers. It does not require either a central server having the global view of the overlay network, as in Piazza [4, 9] or in [16].

The recent work around the coDB peer data management system [17] supports dynamic networks but the first step of the distributed algorithm is to let each node know the network topology. In contrast, in SomeWhere no node does not have to know the topology of the network.

The Kadop system [18] is an infastructure based on distributed hash tables for constructing and querying peer-to-peer warehouses of XML resources semantically enriched by taxonomies and mappings. The mappings that are considered are simple inclusion statement between atomic classes. Compared to KadoP (and also to DRAGO [19]), the mapping language that is dealt with in SomeWhere

is more expressive than simple inclusion statements between atomic classes. It is an important difference which makes SomeWhere able to *combine* elements of answers coming from different sources for answering a query, which KadoP or DRAGO cannot do.

SomeWhere implements in a simpler setting the vision of peer-to-peer data management systems proposed in [20] for relational databases.

6 Conclusion and future work

We have presented the SomeWhere semantic peer-to-peer data management system. Its data model is based on the propositional fragment of the Ontology Web Language recommended by W3C. SomeWhere implements a fully peer-to-peer approach. We have conducted a significant experimentation on networks of 1000 peers. It is presented in [21]. To the best of our knowledge, this is the first experimental study on such large peer-to-peer data management systems. The motivations of this experimentation was twofold. First, to study how deep and how wide reasoning spreads on the network. Second, to evaluate the time needed to obtain answers and to check to what extent SomeWhere is able to support the traffic load.

SomeWhere is the basis of the MediaD project with France Télécom, which aims at enriching peer-to-peer web applications (e.g., Someone [3]) with reasoning services.

We plan to extend SomeWhere in three directions.

We first plan to tackle the problem of possible inconsistency of the distributed schema which can occur because of the mappings, even if the local theories are all consistent. In principle, our algorithm is able to check whether adding a new theory and set of mappings to a consistent SomeWhere network of theories leads to an inconsistency. Therefore, we could forbid a new peer to join the network if it makes the global schema inconsistent, and thus guarantee by contruction that query processing applies on consistent SomeWhere networks. However, this solution is probably too rigid and restrictive to be accepted in practice by users who want to join a SomeWhere network. At least, a new peer whose join leads to an inconsistency would like to know with which other peer(s) its ontology is inconsistent. The problem of detecting the causes of an inconsistency is not trivial and has been extensively studied for centralized theories or knowledge bases. We need to investigate that issue in the SomeWhere distributed setting. We could also decide not to correct the inconsistency but to confine it and answer queries within consistent sub-networks.

Second, we want to extend the SomeWhere data model with binary relations. We are currently exhibiting another propositional transfert for peers relying on the RDF/RDFS data model and accepting conjunctive queries.

Finally, we plan to plug SomeWhere onto a Chord infrastructure [22] in order to make SomeWhere more robust to frequent changes in the network due to peers joins and leaves. In addition, the look-up service offered by Chord could be exploited for optimization purposes of the current SomeWhere query processing.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284** (2001) 35–43 Essay about the possibilities of the semantic web.
2. Rousset, M.C.: Small can be beautiful in the semantic web. In: ISWC 2004, International Semantic Web Conference. (2004)
3. Plu, M., Bellec, P., Agosto, L., van de Velde, W.: The web of people: A dual view on the WWW. In: Int. World Wide Web Conf. (2003)
4. Halevy, A., Ives, Z., Suciu, D., Tatarinov, I.: Schema mediation in peer data management systems. In: ICDE'03. (2003)
5. Ooi, B., Shu, Y., Tan, K.L.: Relational data sharing in peer data management systems. **23** (2003)
6. Arenas, M., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R., Mylopoulos, J. In: The Hyperion project: From data integration to data coordination. (2003)
7. Bernstein, P., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihraheu, I.: Data management for p2p computing: A vision. In: WebDB. (2002)
8. Nedjl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., al.: Edutella: a p2p networking infrastructure based on rdf. In: WWW'02. (2002)
9. Halevy, A., Ives, Z., Tatarinov, I., Mork, P.: Piazza: data management infrastructure for semantic web applications. In: WWW'03. (2003)
10. Halevy, A.Y. In: Logic-based techniques in data integration. Kluwer Academic Publishers (2000) 575–595
11. Madhavan, J., Halevy, A.: Composing mappings among data sources. In: VLDB 03. (2003)
12. Tatarinov, I., Halevy, A.: Efficient query reformulation in peer data management systems. In: SIGMOD 04. (2004)
13. Goasdoué, F., Rousset, M.C.: Answering queries using views. ACM Journal - Transactions on Internet Technology (TOIT) **4** (2004)
14. Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.C., Simon, L.: Distributed reasoning in a p2p setting, short paper. In: ECAI. (2004) 945–946
15. Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.C., Simon, L.: Distributed reasoning in a p2p setting. Technical report, http://www.lri.fr/~goasdoue/bib/ACGRS-TR-1385.pdf (2004)
16. Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Logical fondation of p2p data integration. In: PODS, Paris, France (2004)
17. Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I.: Queries and updates in the codb p2p database system. In: VLDB 2004. (2004)
18. Abiteboul, S., Manolescu, I., Preda, N.: Constructing and querying p2p warehouses of xml resources. In: Workshop on Semantic Web and Databases. (2004)
19. Serafini, L., Tamilin, A.: Drago: Distributed reasoning architecture for the semantic web. Technical report, ITC-IRST (2004)
20. Bernstein, P., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for p2p computing: a vision. In: Proceedings of WebDB 2002. (2002)
21. Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.C., Simon, L.: Scalability study of p2p consequence finding. In: IJCAI, IJCAI (2005)
22. Stoica, I., Morris, R., Karger, D., Kaasshoek, M., Balakrishnan, H.: Chord: a scalable p2p lookup service for internet applications. In: Conference on applications, technologies, archtecture and protocols for computer communications. (2001)