Research online Publishing
www.ronpub.com

Open Access

Open Journal of Web Technologies (OJWT)
Volume 3, Issue 1, 2016

http://www.ronpub.com/ojwt
ISSN 2199-188X

Query Processing in a P2P Network of Taxonomy-based Information Sources

Carlo Meghini [A], Anastasia Analyti [B]

[A] Consiglio Nazionale delle Ricerche, Istituto della Scienza e delle Tecnologie della Informazione, Via G.Moruzzi 1, 56124, Pisa, Italy, carlo.meghini@cnr.it

[B] Institute of Computer Science, Foundation for Research, and Technology – Hellas (FORTH-ICS), P.O. Box. 1385, Heraklion 71110, Crete, Greece, analyti@ics.forth.gr

Abstract

In this study we address the problem of answering queries over a peer-to-peer system of taxonomy-based sources. A taxonomy states subsumption relationships between negation-free DNF formulas on terms and negation-free conjunctions of terms. To the end of laying the foundations of our study, we first consider the centralized case, deriving the complexity of the decision problem and of query evaluation. We conclude by presenting an algorithm that is efficient in data complexity and is based on hypergraphs. We then move to the distributed case, and introduce a logical model of a network of taxonomy-based sources. On such network, a distributed version of the centralized algorithm is then presented, based on a message passing paradigm, and its correctness is proved. We finally discuss optimization issues, and relate our work to the literature.

Type of Paper and Keywords

Regular research paper: *query answering, peer-to-peer systems, P2P, taxonomy-based sources, caching, optimization*

1 Introduction

This paper contributes to the proposal of a logic-based framework for modeling Peer-to-Peer (P2P) networks. Each peer joining a P2P network uses a set of mapping rules, i.e. correspondences to a set of reachable peers to both provide or import data. In the context of a P2P network, let us introduce our framework. Consider an information source $\mathcal{S}$ structured as a tetrad $\mathcal{S} = (T, \preceq, Obj, I)$, where T is a set of terms, $\preceq$ is a taxonomy over concepts expressed using T (e.g. $(\texttt{Animal} \wedge \texttt{FlyingObject}) \vee \texttt{Penguin} \preceq \texttt{Bird}$), Obj is a set of objects and I is the interpretation, that is a function from T to $\mathcal{P}(Obj)$, assigning an extension (*i.e.*, a set of objects) to each term. Now assume that there is a set $\mathcal{N}$ of such sources $\mathcal{N} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, all sharing the same set of objects Obj and related by taxonomic relationships amongst concepts of different sources. These relationships are called *articulations* and aim at bridging the inevitable naming, granularity and contextual heterogeneities that may exist between the taxonomies of the sources (for some examples see [34]). For example, the taxonomy of a peer $\mathcal{S}_1$ could be the following: $\{$ $\texttt{Penguin} \preceq \texttt{Animal}$, $\texttt{Pelican} \preceq \texttt{Animal}$, $\texttt{Ostrich} \preceq \texttt{Animal}$, $(\texttt{Animal} \wedge \texttt{FlyingObject}) \vee \texttt{Penguin} \vee \texttt{Ostrich} \preceq \texttt{Bird}$ $\}$. The object base of $\mathcal{S}_1$ could be the following: $\{$ $\texttt{Ostrich}(1)$, $\texttt{Bird}(2)$, $\texttt{Animal}(3)$, $\texttt{FlyingObject}(3)$ $\}$. $\mathcal{S}_1$ could have an articulation to a peer $\mathcal{S}_2$ like $\{$ $\Pi\iota\nu\gamma\kappa o\upsilon\acute{\iota}\nu o\varsigma_2 \preceq \texttt{Penguin}$, $\Pi\epsilon\lambda\epsilon\kappa\acute{\alpha}\nu o\varsigma_2 \preceq \texttt{Pelican}$ $\}$, an articulation to a peer $\mathcal{S}_3$ like { $\texttt{Animale}_3 \wedge \texttt{Alato}_3 \preceq \texttt{Birds}$ }, and an articulation to two peers $\mathcal{S}_4, \mathcal{S}_5$ of the form: { $(\texttt{Fliegendes Tier}_4) \vee (\texttt{Animal}_5 \wedge \texttt{Volant}_5) \preceq (\texttt{Animal} \wedge \texttt{FlyingObject})$ }.

Network of sources of this kind are nowadays commonplace. For instance, the objects may be web pages, and a source may be a portal serving a specific community endowed with a vocabulary used for indexing web pages. The objects may be library resources such as books, serials, or reports, and a source may be a library describing the content of the resources according to a local vocabulary. The objects may be a category of commercial items, such as cars, and a source may be an e-commerce site which sells the items. And so on. Articulations may be drawn from language dictionaries, or may be the result of cooperation agreements, such as in the case of sources belonging to the same organization. In certain cases, articulations can be constructed automatically, for instance using the data-driven method proposed in [32].

In this paper we address the problem of answering Boolean queries over a peer-to-peer (P2P) system of this kind of sources.

The case of fully heterogeneous conceptual models makes uniform global access extremely challenging. This is the case that we are interested in. From a data modeling point of view several approaches for P2P systems have been proposed, including relational-based approaches [5], XML-based approaches [23] and RDF-based [28]. In this paper we consider the fully heterogeneous conceptual model approach (where each peer can have its own schema), with the only restriction that each conceptual model is represented as a taxonomy. A taxonomy can range from a simple tree-structured hierarchy of terms, to the concept lattice derived by Formal Concept Analysis [21], or to the concept lattice of a Description Logics theory.

As a specific example, in the context of Linked Open Data [26], consider peers each holding (i) an RDF ontology [15], a set of taxonomic relationships between local classes, and (iii) a set of articulations between local classes and classes of the local RDF ontology and/or other RDF ontologies. Then, the instances of the local classes are enriched by the instances of the local classes and the instances of the classes of other RDF ontologies, according to the particular relationships. Consider for instance that the concepts of the peers in the introductory example are classes of peer RDF ontologies and the articulations along with the local taxonomies are relationships between these classes. In this case, a semi-naive bottom-up evaluation algorithm can be applied to the RDF inference rules provided in [25] in order to compute the closure of each local RDF ontology. This way the local classes are filled with local objects. Then, our algorithms can be straightforwardly applied for finding querying class instances based on all relationships.

In particular:

- we analyze the theoretical aspects of query evaluation against a source, and an algorithm is derived which extends a hypergraph-based method for satisfiability of propositional Horn clauses. The algorithm is conceptually very simple and has polynomial time complexity with respect to the size of Obj.
- we present a (asynchronous messaging) distributed query evaluation procedure, based on a functional model of a peer; correctness and complexity of this procedure are given;
- we describe several optimization techniques that can be used for improving the efficiency of query evaluation based on caching;
- we relate our work to the existing literature on peer-to-peer systems.

Some parts of the work reported in this paper have been already published. Namely, [34] presents a first model of a network of articulated sources, while [33] studies query evaluation on taxonomies including only term-to-term subsumption relationships. Finally, [27] presents a procedure for evaluating queries over centralized sources supporting term-to-query subsumption relationships, as well as hardness results for extensions. With respect to [34, 33, 27], this paper improves the theoretical aspects for query evaluation over the centralized case. Additionally, it provides distributed query evaluation algorithms for the case that term-to-query subsumption relationships are considered along with their optimizations which are based on caching.

The paper is structured as follows: Section 2 introduces sources, presenting the centralized query evaluation procedure. Networks of sources are considered in Section 3, where our algorithm for query evaluation on networks is presented, and Section 4 discusses optimization issues. Section 5 compares our work with related work and Section 6 concludes the paper.

2 Foundations

This Section defines information sources and the query evaluation problem. The algorithmic foundations of this problem are given and an efficient query evaluation method is provided. These results will be applied later, upon studying networks of sources.

2.1 The Model

The basic notion of the model is that of *terminology:* a terminology T is a non-empty set of terms. From a terminology, *queries* can be defined.

Definition 1 (Query): The *query language* associated to a terminology T, $\mathcal{L}_T$, is the language defined by the following grammar, where t is a term of T :

$$\begin{aligned} q &::= d \mid q \vee d \\ d &::= t \mid t \wedge d. \end{aligned}$$

An instance of q is called a *query,* while an instance of d is called a *conjunctive query* and a *disjunct* of q whenever d occurs in q. □

Terms and conjunctive queries can be used for defining taxonomies.

Definition 2 (Taxonomy): A *taxonomy* over a terminology T is a pair $(T, \preceq)$ where $\preceq$ is any set of pairs (q, d) where q is any query and d is a conjunctive query. □

For example, if $T = \{a1, a2, b1, b2, b3, c1\}$ then a taxonomy over T could be $(T, \preceq)$ where (using an infix notation) $\{(b1 \wedge b2) \vee b3 \ \preceq \ a1 \wedge a2,\ a1 \wedge a2 \ \preceq \ c1\}$.

If $(q, q') \in \preceq$, we say that q is subsumed by q' and we write $q \preceq q'$.

Definition 3 (Interpretation): An *interpretation* for a terminology T is a pair (Obj, I), where Obj is a finite, non-empty set of objects and I is a total function assigning a possibly empty set of objects to each term in T, *i.e.* $I : T \to \mathcal{P}(Obj)$. □

Interpretations are used to define the semantics of the query language:

Definition 4 (Query extension): Given an interpretation I of a terminology T and a query $q \in \mathcal{L}_T$, the *extension of q in I,* q^I, is defined as follows:

1. $(q \vee d)^I = q^I \cup d^I$
2. $(d \wedge t)^I = d^I \cap t^I$
3. $t^I = I(t)$. □

Since $\cdot^I$ is an extension of the interpretation function I, we will simplify notation and will write $I(q)$ in place of q^I. We can now define an *information source* (or simply *source*).

Definition 5 (Information source): An *information source* S is a 4-tuple $S = (T_S, \preceq_S, Obj_S, I_S)$, where $(T_S, \preceq_S)$ is a taxonomy and (Obj_S, I_S) is an interpretation for T_S. □

When no ambiguity will arise, we will omit the subscript in the components of sources and equate I with (Obj, I), for simplicity. Moreover, given a source $S = (T, \preceq, Obj, I)$ and an object $o \in Obj$, the *index of o in S*, $ind_S(o)$, is given by the terms in whose interpretation o belongs, *i.e.*:

$$ind_S(o) = \{t \in T \mid o \in I(t)\}.$$

The interpretations that reflect the semantics of subsumption are as customary called *models,* defined next.

Definition 6 (Models of a source): Given two interpretations I, I' of the same terminology T,

1. I is a *model* of the taxonomy $(T, \preceq)$ if $q \preceq q'$ implies $I(q) \subseteq I(q')$;
2. I is smaller than I', $I \leq I'$, if $I(t) \subseteq I'(t)$ for each term $t \in T$;
3. I is a *model* of a source $S = (T, \preceq, Obj, I')$ if it is a model of $(T, \preceq)$ and $I' \leq I$. □

Based on the notion of model, the answer to a query is finally defined.

Definition 7 (Answer): Given a source $S = (T, \preceq, Obj, I)$ and a query $q \in \mathcal{L}_T$, the *answer of q in S,* $ans(q, S)$, is given by $ans(q, S) = \{o \in Obj \mid o \in J(q)$ for all models J of $S\}$. □

Since we are exclusively interested in query evaluation, we can restrict ourselves to simpler notions of sources and queries, which are equivalent to those defined so far from the answer point of view. To begin with, we observe that a pair (q, q') in a taxonomy is interpreted (in Definition 6 point 1) as an implication $q \to q'$. Now, by a simple truth table argument, it can be easily verified that the propositional formula:

$$(C_1 \vee \ldots \vee C_n) \to (t_1 \wedge \ldots \wedge t_m)$$

where each C_i is any propositional formula, is logically equivalent to the formula:

$$(C_1 \to t_1) \wedge (C_1 \to t_2) \wedge \ldots \wedge (C_1 \to t_m) \wedge \ldots \wedge \\ (C_n \to t_1) \wedge (C_n \to t_2) \wedge \ldots \wedge (C_n \to t_m),$$

in that the two formulae have the same models. Based on this equivalence, the *simplification* of a taxonomy $(T, \preceq)$ is defined as the taxonomy $(T, \preceq^s)$, where:

$$\preceq^s = \{(C, t) \mid (C_1 \vee \ldots \vee C_n) \ \preceq \ (t_1 \wedge \ldots \wedge t_m),\ C \in \{C_1, \ldots, C_n\},\ t \in \{t_1, \ldots, t_m\}\}.$$

Correspondingly, the simplification of a source $S = (T, \preceq, Obj, I)$ is defined to be the source $S^s = (T, \preceq^s, Obj, I)$. It is not difficult to see that:

Proposition 1: J is a model of a source S if and only if it is a model of S^s. □

The simplification of the taxonomy in the previous example is given by:
$\{(b1 \wedge b2) \preceq^s a1,\ (b1 \wedge b2) \preceq^s a2,\ b3 \preceq^s a1,\ b3 \preceq^s a2,\ b3 \preceq^s a2,\ a1 \wedge a2 \preceq^s c1\}$.

For simplicity, from now on $\preceq$ and S will stand for $\preceq^s$ and S^s, respectively.

Finally, non-term queries can be replaced by term queries by inserting appropriate relationships into the taxonomy. Specifically:

Proposition 2: For all sources $S = (T, \preceq, Obj, I)$ and non-term queries $q \in \mathcal{L}_T$, let t_q be any term not in T and moreover

$$\begin{aligned} T^q &= T \cup \{t_q\} \\ \preceq^q &= \preceq \cup \{(t_1 \wedge \ldots \wedge t_m, t_q) \mid t_1 \wedge \ldots \wedge \\ &\quad\ t_m \text{ is a disjunct of } q\} \\ I^q &= I \cup \{(t_q, \emptyset)\}. \end{aligned}$$

Then, $ans(q, S) = ans(t_q, S^q)$ where $S^q = (T^q, \preceq^q, Obj, I^q)$. □

In practice, the terminology T^q includes one additional term t_q, which has an empty interpretation and subsumes each query disjunct $t_1 \wedge \ldots \wedge t_m$. The size of S^q is clearly polynomial in the size of S and q.

For example, assume that the query $q = (t_1^1 \wedge \ldots \wedge t_{m_1}^1) \vee (t_1^2 \wedge \ldots \wedge t_{m_2}^2)$ is posed to a source S. If the relation $\preceq$ of S is extended by $t_1^1 \wedge \ldots \wedge t_{m_1}^1 \preceq t_q$ and $t_1^2 \wedge \ldots \wedge t_{m_2}^2 \preceq t_q$, where t_q is a new term then the answer of q w.r.t. S is the same as the answer of t_q in the new source.

In light of the last Proposition, the problem of query evaluation amounts to determine $ans(t, S)$ for given term t and source S, while the corresponding decision problem consists in checking whether $o \in ans(t, S)$, for a given object o.

2.2 The Decision Problem

Given a source $S = (T, \preceq, Obj, I)$, $o \in Obj$, and $t \in T$, the decision problem $o \in ans(t, S)$ has an equivalent formulation in propositional datalog. We define the propositional datalog program P_S as follows:

$$P_S = C_S \cup I_S \cup Q_S$$

where

$$\begin{aligned} C_S &= \{t' \leftarrow t_1, \ldots, t_m \mid (t_1 \wedge \ldots \wedge t_m, t') \in \preceq^r\} \\ I_S &= \{u \leftarrow \mid u \in ind_S(o)\} \\ Q_S &= \{\leftarrow t\} \end{aligned}$$

It is easy to see that:

Lemma 1: For all sources $S = (T, \preceq, Obj, I)$, $o \in Obj$ and $t \in T$, $o \in ans(t, S)$ iff P_S is unsatisfiable. □

Based on Lemma 1, the decision problem $o \in ans(t, S)$ is connected to directed B-hypergraphs, which are introduced next. We will mainly use definitions and results from [20].

A *directed hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of vertexes and $\mathcal{E} = \{E_1, E_2, \ldots, E_m\}$ is the set of directed hyperedges, where $E_i = (\tau(E_i), \chi(E_i))$ with $\tau(E_i), \chi(E_i) \subseteq \mathcal{V}$ for $1 \leq i \leq m$. $\tau(E_i)$ is said to be the *tail* of E_i, while $\chi(E_i)$ is said to be the *head* of E_i. A *directed B-hypergraph* (or simply *B-graph*) is a directed hypergraph, where the head of each hyperedge E_i, denoted as $h(E_i)$, is a single vertex.

A taxonomy can naturally be represented as a B-graph whose hyperedges represent one-to-one the subsumption relationships of the transitive reduction of the taxonomy. In particular, the *taxonomy B-graph* of a taxonomy $(T, \preceq)$ is the B-graph $\mathcal{H} = (T, \mathcal{E}_{\preceq})$, where

$$\mathcal{E}_{\preceq} = \{(\{t_1, \ldots, t_m\}, u) \mid (t_1 \wedge \ldots \wedge t_m, u) \in \preceq^r\}.$$

Figure 1 left presents a taxonomy, whose B-graph is shown in the same Figure right.

A *path* P_{st} of length q in a B-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a sequence of nodes and hyperedges

$$P_{st} = (s = v_1, E_{i_1}, v_2, E_{i_2}, \ldots, E_{i_q}, v_{q+1} = t),$$

where: $s \in \tau(E_{i_1})$, $h(E_{i_q}) = t$ and $h(E_{i_{j-1}}) = v_j \in \tau(E_{i_j})$ for $2 \leq j \leq q$. If P_{st} exists, t is said to be *connected* to s. If $t \in \tau(E_{i_1})$, P_{st} is said to be a *cycle*; if all hyperedges in P_{st} are distinct, P_{st} is said to be *simple*. A simple path is *elementary* if all its vertexes are distinct.

A *B-path* π_{st} in a B-graph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a minimal (with respect to deletion of vertexes and hyperedges) hypergraph $\mathcal{H}_\pi = (\mathcal{V}_\pi, \mathcal{E}_\pi)$, such that:

1. $\mathcal{E}_\pi \subseteq \mathcal{E}$
2. $\{s, t\} \subseteq \mathcal{V}_\pi$
3. $x \in \mathcal{V}_\pi$ and $x \neq s$ imply that x is connected to s in $\mathcal{H}_\pi$ by means of a cycle-free simple path.

Vertex y is said to be *B-connected* to vertex x if a B-path π_{xy} exists in $\mathcal{H}$.

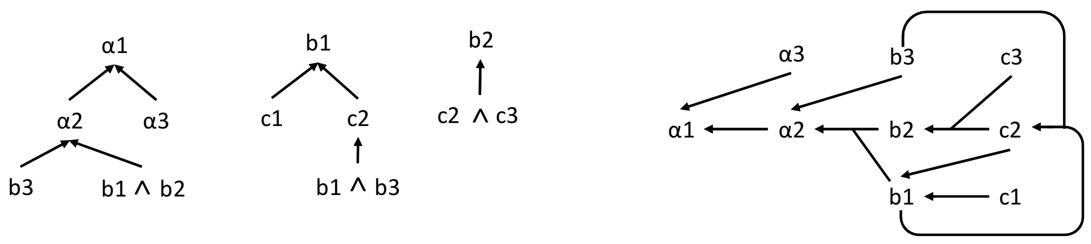

Figure 1: A taxonomy and its B-graph

B-graphs and satisfiability of propositional Horn clauses are strictly related. The B-graph *associated to* a set of Horn clauses has 3 types of directed hyperedges to represent each clause:

- the clause $p \leftarrow q_1 \wedge q_2 \wedge \ldots \wedge q_s$ is represented by the hyperedge $(\{q_1, q_2, \ldots, q_s\}, p)$;
- the clause $\leftarrow q_1 \wedge q_2 \wedge \ldots \wedge q_s$ is represented by the hyperedge $(\{q_1, q_2, \ldots, q_s\}, \mathit{false})$;
- the clause $p \leftarrow$ is represented by the hyperedge $(\{\mathit{true}\}, p)$.

The following result is well-known:

Proposition 3 ([20]): A set of propositional Horn clauses is satisfiable if and only if in the associated B-graph, *false* is not B-connected to *true*. □

We now proceed to show the role played by B-connection in query evaluation. For a source $S = (T, \preceq, Obj, I)$ and an object $o \in Obj$, the *object decision graph* (simply the *object graph*) is the B-graph $\mathcal{H}_o = (T, \mathcal{E}_o)$, where

$$\mathcal{E}_o = \mathcal{E}_{\preceq} \cup \bigcup \{(\{\mathit{true}\}, u) \mid u \in ind_S(o)\}.$$

Figure 2 presents the object graph for the taxonomy shown in Figure 1 and an object o such that $ind_S(o) = \{c1, c2, c3\}$.

We can now prove:

Proposition 4: For all sources $S = (T, \preceq, Obj, I)$, terms $t \in T$, and objects $o \in Obj$, $o \in ans(t, S)$ iff t is B-connected to *true* in the object graph $\mathcal{H}_o$.
□

Proof of Proposition 4: From Lemma 1, $o \in ans(t, S)$ iff P_S is unsatisfiable iff (by Proposition 3) *false* is B-connected to *true* in the associated B-graph. By construction, $\mathcal{H}_o$ is the B-graph associated to P_S, where t plays the role of *false*.

2.3 An Algorithm for Query Evaluation

A typical approach for query evaluation is resolution, used also in peer-to-peer networks [3, 4, 2]. Here, we propose a simpler method to perform query evaluation, based on B-graphs. Our method relies on the following result, which is just a re-phrasing of Proposition 4:

Corollary 1: For all sources $S = (T, \preceq, Obj, I)$, $o \in Obj$ and term queries $t \in T$, $o \in ans(t, S)$ if and only if either $o \in I(t)$ or there exists a hyperedge $(\{u_1, \ldots, u_r\}, t) \in \mathcal{E}_{\preceq}$ such that $o \in \bigcap\{ans(u_i, S) \mid 1 \leq i \leq r\}$. □

This corollary simply "breaks down" Proposition 4 based on the distance between t and *true* in the object graph $\mathcal{H}_o$. If $o \in I(t)$, then $t \in ind_S(o)$, hence there is a hyperedge (in fact, a simple arc) from *true* to t in $\mathcal{H}_o$, which are 1 hyperedge distant from each other. If $o \notin I(t)$, then there are at least two hyperedges in between *true* and t. Let us assume that h is the one whose head is t. Since t is B-connected to *true*, each term u_i in the tail of h is B-connected to *true*. But this simply means, again by Proposition 4, that $o \in ans(u_i, S)$ for all the terms u_i, and so we have the forward direction of the Corollary. The backward direction of the Corollary is straightforward. Notice that, by point 3 in the definition of B-path, t is connected to each u_i by a cycle-free simple path; this fact is used by the procedure QE in order to correctly terminate in presence of loops in the taxonomy B-graph $\mathcal{H}$.

The procedure QE, presented in Figure 3, computes $ans(t, S)$ for a given term t (and an implicitly given source S) by applying in a straightforward way Corollary 1. To this end, QE must be invoked as QE$(t, \{t\})$. The second input parameter of QE is the set of terms on the *path* from t to the currently considered term x. This set is used to guarantee that t is connected to all terms considered in the recursion by a cycle-free simple path. QE accumulates in R the result. The correctness of QE can be established by just observing that, for all objects $o \in Obj$, o is in the set R returned by QE$(t, \{t\})$ if and only if o satisfies the two conditions expressed by Corollary 1.

As an example, let us consider the sequence of calls made by the procedure QE in evaluating the

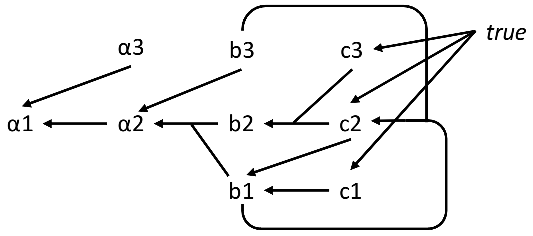

Figure 2: An object graph

```
QE($x$ : **term** ; $A$ : **set of terms**);
1. $R \leftarrow I(x)$
2. **for each** hyperedge $\langle \{u_1, \ldots, u_r\}, x \rangle$ in $\mathcal{H}$ **do**
3.    **if** $\{u_1, \ldots, u_r\} \cap A = \emptyset$ **then** $R \leftarrow R \cup (\mathrm{QE}(u_1, A \cup \{u_1\}) \cap \ldots \cap \mathrm{QE}(u_r, A \cup \{u_r\}))$
4. **return**($R$)
```

Figure 3: The procedure QE

Table 1: Evaluation of QE$(a2, \{a2\})$

Call	Result
$\mathrm{QE}(a2, \{a2\})$	$I(a2) \cup \mathrm{QE}(b3, \{a2, b3\}) \cup (\mathrm{QE}(b1, \{a2, b1\}) \cap \mathrm{QE}(b2, \{a2, b2\}))$
$\mathrm{QE}(b3, \{a2, b3\})$	$I(b3)$
$\mathrm{QE}(b1, \{a2, b1\})$	$I(b1) \cup \mathrm{QE}(c1, \{a2, b1, c1\}) \cup \mathrm{QE}(c2, \{a2, b1, c2\})$
$\mathrm{QE}(b2, \{a2, b2\})$	$I(b2) \cup (\mathrm{QE}(c2, \{a2, b2, c2\}) \cap \mathrm{QE}(c3, \{a2, b2, c3\}))$
$\mathrm{QE}(c1, \{a2, b1, c1\})$	$I(c1)$
$\mathrm{QE}(c2, \{a2, b1, c2\})$	$I(c2) \star$
$\mathrm{QE}(c2, \{a2, b2, c2\})$	$I(c2) \cup (\mathrm{QE}(b1, \{a2, b2, c2, b1\}) \cap \mathrm{QE}(b3, \{a2, b2, c2, b3\}))$
$\mathrm{QE}(c3, \{a2, b2, c3\}))$	$I(c3)$
$\mathrm{QE}(b1, \{a2, b2, c2, b1\})$	$I(b1) \cup \mathrm{QE}(c1, \{a2, b2, c2, b1, c1\}) \star$
$\mathrm{QE}(b3, \{a2, b2, c2, b3\}))$	$I(b3)$
$\mathrm{QE}(c1, \{a2, b2, c2, b1, c1\})$	$I(c1)$

query $a2$ in the example source of Figure 1, as shown in Table 1. The calls marked with a $\star$ are those in which the test in line 3 gives a negative result. Upon evaluating $\text{Q}\textsc{e}(c2, \{a2, b1, c2\})$ the procedure realizes that the only incoming hyperedge in $c2$ is $\langle\{b1, b3\}, c2\rangle$, whose tail $\{b1, b3\}$ has a non-empty intersection with the current path $\{a2, b1, c2\}$; so the hyperedge is ignored. In this case, the cycle $(b1, c2, b1)$ is detected and properly handled. Analogously, upon evaluating $\text{Q}\textsc{e}(b1, \{a2, b2, c2, b1\})$, the cycle $(c2, b1, c2)$ is detected and properly handled. Also notice the difference between the calls $\text{Q}\textsc{e}(c2, \{a2, b1, c2\})$ and $\text{Q}\textsc{e}(c2, \{a2, b2, c2\})$. They both concern $c2$, but in the former case, $c2$ is encountered upon descending along the path $(a2, b1, c2)$ whose next hyperedge is $\langle\{b1, b3\}, c2\rangle$; following that hyperedge, would lead the computation back to the node $b1$, which has already been met, thus the result of the call is just $I(c2)$. In the latter case, $c2$ is encountered upon descending along the path $(a2, b2, c2)$, thus the hyperedge leading to $b1$ and $b3$ must be followed, since none of the terms in its tail have been touched upon so far.

From a complexity point of view, QE visits all terms that lie on cycle-free simple paths ending at the query term t in the taxonomy B-graph $\mathcal{H}$. Now, it is not difficult to see that these paths may be exponentially many in the size of the taxonomy. As an illustration, let us consider the taxonomy whose B-graph contains the hyperedges of Table 2.

Let us assume t is the query term. It is easy to verify that there are 2^{n-1} cycle-free simple paths connecting u_1 to t, one for each sequence of the form

$$(u_1\ f_1\ x_2\ f_2\ \ldots\ x_{n-1}\ f_{n-1}\ x_n\ h_n\ t)$$

where f_i can be either h_i (in which case x_{i+1} is u_{i+1}) or g_i (in which case x_{i+1} is v_{i+1}) for $1 \le i \le n-1$.

On the other hand, for each query term, QE performs set-theoretic operations on sets of objects, which initially are interpretations of terms. Thus, we conclude that QE has polynomial time complexity w.r.t. the size of Obj.

3 Networks of Information Sources

In this Section we introduce networks of information sources. The model is first outlined, and then query evaluation is considered.

3.1 The Model

In order to be a component of a networked information system, a source is endowed with additional subsumption relations, called articulations, which relate the source terminology to the terminologies of other sources of the same kind.

Definition 8 (Articulation): Given two terminologies T and U, an *articulation* from T to U, $\preceq_{TU}$, is a non-empty binary relation from $\mathcal{L}_U$ to T, such that $q \preceq_{TU} t$ implies that q is a conjunctive query. □

An articulation relationship is not syntactically different from a subsumption relationship, except that its head may be a term of a different terminology than the one where the terms making up its tail come from.

Definition 9 (Articulated Source): An *articulated source* $\mathcal{S}$ over $k \ge 0$ disjoint terminologies $T_1, ..., T_k$, is a 5-tuple $\mathcal{S} = (T_\mathcal{S}, \preceq_\mathcal{S}, Obj, I_\mathcal{S}, R_\mathcal{S})$, where:

- $(T_\mathcal{S}, \preceq_\mathcal{S}, Obj, I_\mathcal{S})$ is a source;
- $R_\mathcal{S}$ is a set of articulations $R_\mathcal{S} = \{\preceq_{T_\mathcal{S},T_1}, \ldots, \preceq_{T_\mathcal{S},T_k}\}$. □

Articulations are used to connect an articulated source to other articulated sources, so creating a networked information system. An articulated source $\mathcal{S}$ with an empty stored interpretation, *i.e.* $I_\mathcal{S}(t) = \emptyset$ for all $t \in T_\mathcal{S}$, is called a *mediator* in the literature.

Definition 10 (Network): A *network of articulated sources*, or simply a *network*, $\mathcal{N}$ is a non-empty set of articulated sources $\mathcal{N} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, where each $\mathcal{S}_i$ is articulated over the terminologies of some of the other sources in $\mathcal{N}$ and all terminologies $T_{\mathcal{S}_1}, \ldots, T_{\mathcal{S}_n}$ of the sources in $\mathcal{N}$ are disjoint. □

Notice that the domain of the interpretation of an articulated source is independent from the source, thus the same for any articulated source. This is not necessary for our model to work, just reflects a typical situation of networked resources such as URLs. Relaxing this constrain would have no impact on the results reported in the present study.

Since in a network: (a) there is no source acting at the global level, (b) all sources store data, and (c) as we will see, data are exchanged via direct communication, each source can be seen as, and will in fact be called, a *peer*, and the network as a *peer-to-peer* information system. Articulations of the network peers will also be referred as *P2P mappings*.

An intuitive way of interpreting a network is to view it as a single source which is distributed along the nodes of a network, each node dealing with a specific vocabulary. The global source can be logically constructed by removing the barriers which separate local sources, as if (virtually) collecting all the network information in a single repository. The notion of *network source* captures this interpretation of a network.

Table 2: Example hyperedges

$h_1 : (\{u_1, v_1\}, u_2)$	$h_2 : (\{u_2, v_2\}, u_3)$	$\dots$	$h_{n-1} : (\{u_{n-1}, v_{n-1}\}, u_n)$	$h_n : (\{u_n, v_n\}, t)$
$g_1 : (\{u_1, v_1\}, v_2)$	$g_2 : (\{u_2, v_2\}, v_3)$	$\dots$	$g_{n-1} : (\{u_{n-1}, v_{n-1}\}, v_n)$	

Definition 11 (Network source): The *network source* $S_{\mathcal{N}}$ of a network of articulated sources $\mathcal{N} = \{\mathcal{S}_1, \dots, \mathcal{S}_n\}$, is the source $S_{\mathcal{N}} = (T_{\mathcal{N}}, \sqsubseteq, Obj, I_{\mathcal{N}})$, where:

- $T_{\mathcal{N}} = \bigcup_{i=1}^{n} T_{\mathcal{S}_i}$;
- $I_{\mathcal{N}} = \bigcup_{i=1}^{n} I_{\mathcal{S}_i}$
- $\sqsubseteq = (\bigcup_{i=1}^{n} \sqsubseteq_{\mathcal{S}_i})^*$

where $\sqsubseteq_{\mathcal{S}_i}$ is the *total subsumption* of the source $\mathcal{S}_i$, given by the union of the subsumption relation $\preceq_{\mathcal{S}_i}$ with all articulations of the source, that is:

$$\sqsubseteq_{\mathcal{S}_i} = \preceq_{\mathcal{S}_i} \cup \bigcup R_{\mathcal{S}_i}$$

and A^* denotes the transitive closure of the binary relation A. A *network query* is a query over $T_{\mathcal{N}}$. □

It is not difficult to see that $\sqsubseteq$ is reflexive and transitive, and every non-trivial subsumption relationship in it relates a conjunctive query in anyone of the terminologies $T_{\mathcal{S}_1}, \dots, T_{\mathcal{S}_n}$ to a single term. Thus, $S_{\mathcal{N}}$ is indeed a source. Such source emerges in a bottom-up manner from the articulations of the peers. This distinguishes peer-to-peer systems from federated distributed databases.

A *network query* q is a query in anyone of the query languages supported by the network, that is $q \in \mathcal{L}_{T_{\mathcal{S}_i}}$ for some $i \in [1, n]$. As it will be evident, the method that we will set up only requires minor modifications to be able to evaluate also queries in the language $\mathcal{L}_{T_{\mathcal{N}}}$, that is queries that mix terms from different terminologies. We do not provide this facility because it does not seem to make much sense in our vision.

The answer to a network query q, or *network answer,* is given by $ans(q, S_{\mathcal{N}})$.

Figure 4 presents the taxonomy of a network source $S_{\mathcal{N}}$, where $\mathcal{N}$ consists of 3 peers $\mathcal{N} = \{P_a, P_b, P_c\}$. As it can be verified, this is the same taxonomy as the one shown in Figure 1, except that now some of its subsumption relationships are elements of articulations.

3.2 Network Query Evaluation

This Section presents a network query evaluation procedure based on the method devised in the centralized case. First, a functional model of each peer is introduced, then the algorithms corresponding to the operations on the interface of the peer are given. Correctness and complexity of these algorithms are discussed in Section 3.3, while Section 4 concludes by considering optimization issues.

3.2.1 The Functional Model of a Peer

In order to illustrate our query evaluation procedure, we now define a peer from a functional point of view. In this respect, we see a peer as a software component uniquely identified in the network by a peer ID. The interface of a peer exposes just one method:

- QUERY, which takes as input a network query q and evaluates it, returning the set of objects $ans(q, S_{\mathcal{N}})$.

The user (whether human or application program) is supposed to use this method for the evaluation of network queries. We assume that q is expressed in the query language of the peer. As it will be argued in due course, this assumption can be relaxed without any substantial change to our framework.

In addition to QUERY, a peer has methods for sending to or receiving messages from other peers. We do not enter into the details of these methods: there are several options, which do not make any difference from the point of view of our model. Instead, we detail the types of messages that can be exchanged between peers. These can be of one of the following 2 types:

- ASK: by sending a message of this kind to a peer P, the present peer asks P to evaluate a term query on P's query language. The receiving peer P processes ASK messages according to the QE procedure (Figure 3), as we will see in detail below. An ASK message has the following fields:
  - *PID*: the id of the present peer, which is sending the message;
  - *QID*: the id of the query that *PID* is sending for evaluation;
  - *t:* the query term of *QID*;
  - *A:* the set of already visited terms. These two last parameters are those of the QE procedure.
- TELL: by sending a message of this kind to a peer P, the present peer returns to P the result of the evaluation of a term query which had previously been ASK-ed by P. A TELL message has the following fields:

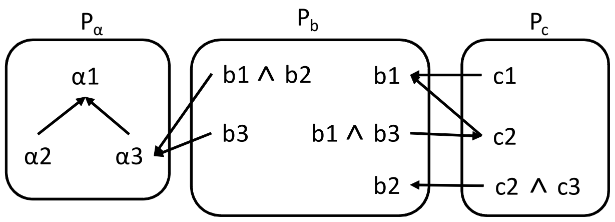

Figure 4: A network taxonomy

- *QID*: the ID of the query whose result is being returned;
- *RES:* the set of objects resulting from the evaluation of *QID*.

We will denote the sending of a message of one of these two kinds m to the peer P as P **:** *m(field values)*. By decoupling the request of evaluation from the return of the result, we aim at minimizing the number of sessions open at any time between peers, thus removing a serious obstacle towards scalability. QUERY does not follow this paradigm since it involves only a local interaction.

Each peer processes the incoming messages depending on their type and content. In order to carry out this work, the peer keeps a *(query) log,* that is a set of objects, each associated to a query in whose evaluation the peer is currently involved. A log object has the following attributes:

- *PID*: the id of the peer who sent the query (can be the local peer itself);
- *QID*: the id of the query;
- *t:* the query term (we recall that we need to deal only with term queries);
- *n:* the number of open calls in *QID* (see next paragraph);
- *QP*: the query program representing the current status of evaluation of *QID*. A query program is a set of *sub-programs* $\{SP_1, \ldots, SP_k\}$ where each sub-program SP_j is a set of *calls.* A call is a sub-query of *QID*, and can be:
  - *open,* meaning that the sub-query is being evaluated, in which case the call is the sub-query id; or
  - *closed,* meaning the sub-query has been evaluated, in which case the call is the resulting set of objects.

Since no two log objects can have the same query id, we will represent a log object as a 5-tuple (*PID*,*QID*,*t*,*n*,*QP*).

```
QUERY($q$ : **query**);
1.  $t \leftarrow$ MODIFY-TAXONOMY($q$)
2.  ID $\leftarrow$ NEW-QUERY-ID
3.  self: ASK(self, ID, $t$, $\{t\}$)
4.  wait until ID is closed then
5.  (PID, QID, $t$, $n$, $R$) $\leftarrow$ DELETE(ID)
6.  CLEANUP-TAXONOMY($t$)
7.  return($R$)
```

Figure 5: The QUERY procedure

3.2.2 QUERY

Let us assume that the input query q posed to a peer $\mathcal{S}$, is given by

$$q = \bigvee C_i$$

where each C_i is a conjunctive query. As a first step, QUERY reduces q to a term query t by generating a new term t not in $T_{\mathcal{N}}$ and inserting a new hyperedge (C_i, t) into the local taxonomy B-graph (i.e. that corresponding to $(T_{\mathcal{S}}, \sqsubseteq_{\mathcal{S}})$), for each conjunctive query C_i in q. This work is carried out by the function MODIFY-TAXONOMY, which returns the newly generated term t. A new query id for t is subsequently obtained by QUERY, and an ASK message is sent to the peer itself for evaluating t. As required by QE, the set of already visited terms consists just of t itself. At this point QUERY hangs on the log, until the log object associated to the query t is closed, that is the number of its open call is 0. Notice that this object is created only after the ASK message sent on line 3 is processed, but this creates no problem, as all QUERY has to do in the meantime is wait. When the log object is finally closed, QUERY retrieves it and deletes it from the log, by using the function DELETE, which returns the object itself. When the object is closed, its query program, that is the value of the last field, equals to $ans(t, S_{\mathcal{N}})$. This value is assigned to the variable R. On line 6, the subsumption relationships inserted by MODIFY-TAXONOMY are removed by CLEANUP-TAXONOMY, and R is finally returned.

As an example, let us consider the network shown in Figure 4, whose corresponding B-graph is shown in Figure 6, and the query $(a2 \wedge a3)$ on peer P_a. When given as input to QUERY, this query is passed on to MODIFY-TAXONOMY, which adds the hyperedge $(\{a2, a3\}, t)$ to the taxonomy B-graph and returns the newly generated term t. Let us assume that $q1$ is the id of the new query. QUERY then sends the message ASK(P_a, $q1$, t, $\{t\}$) to itself, and gets into the wait loop until the query is evaluated.

3.2.3 ASK

For readability, we will describe ASK and TELL as if they were methods whose parameters are the message fields. ASK (Figure 7) uses the following variables:

- n : counts how many sub-queries the input query *QID* generates;
- *QP*: is the initial query program of *QID*;
- Q : is a queue holding the information to send the ASK messages required to evaluate *QID*;
- C : is the query sub-program being currently computed.

After initialization, ASK performs (line 2) the same test as QE, looking for a hyperedge h in the local B-graph whose head is the given term t and whose tail is disjoint form A. If no such hyperedge is found, then n remains 0, the test on line 10 fails, and the result of the evaluation of the given term query t is just $I(t)$ (as QE establishes), which ASK returns by sending a TELL message to the invoking peer *PID* (line 15). If instead a hyperedge h is found, then the intersection of the evaluation of each term u_i in its tail should be added to the result, according to QE. In order to achieve the same behavior, ASK enters a loop in which it processes each term u_i to the end of constructing in C the query sub-program associated to h. First, a new query id *ID* is generated (line 5) to denote the sub-query on u_i; the newly generated id is then added to C. On line 7, the number of open calls is increased by one, and on line 8 the required information to evaluate the query u_i is enqueued in Q. This information is:

- the id of the peer P_h holding the terms in the tail of the hyperedge h; we assume this information is stored with the hyperedge just for convenience, the peer can also store it separately;
- the *ID* of the sub-query;
- the query term u_i and
- the set of the visited terms $A \cup \{u_i\}$, as in QE.

Each sub-program so generated is added to *QP*, after considering all relevant hyperedges (line 9). At this point, if the number of open calls is positive, ASK uses the function PERSIST in order to create the log object representing the query *QID*, and to persist it in the log. Once the log object is successfully persisted, ASK must launch the evaluation of the generated sub-queries, which it does in the loop on lines 12-14. Until Q is empty, it dequeues the information for constructing an ASK message for each sub-query, and sends such message to the peer P_h. The value of the first message field is the peer identity (*self*), as the invoking peer.

At this point, it can be easily verified that the assumption that all terms in the tail of a hyperedge are from the same terminology, namely that of peer P_h, *can be relaxed without any impact on the query evaluation procedure.* In logical terms, this is the assumption that the conjunctive queries on the left-hand side of subsumption relationships are from the query language of one peer. We have made this assumption because it fits our vision of a network. But ASK can easily work also with hyperedges whose tails have terms from different terminologies: all that is required is to store the id of the peer holding each term, rather than the id of the peer holding the whole hyperedge.

Let us resume our running example. Upon processing the message (P_a, $q1$, t, $\{t\}$), ASK finds that the hyperedge $h = (\{a2, a3\}, t)$ passes the test on line 2, and enters the loop on the tail of h. For term $a2$, assuming the generated query id is $q2$, the record (P_a, $q2$, $a2$, $\{t, a2\}$) is enqueued in Q, while for term $a3$, (generated id $q3$) it is enqueued the record (P_a, $q3$, $a3$, $\{t, a3\}$). As there are no more hyperedges and $n = 2$, a new log object is created to represent the query t. The attributes of this object are:

- *PID* = P_a
- *QID* = $q1$
- $t = t$
- $n = 2$
- *QP* = $\{\{q2, q3\}\}$.

Now two ASK messages are send to P_a :

1. (P_a, $q2$, $a2$, $\{t, a2\}$), and
2. (P_a, $q3$, $a3$, $\{t, a3\}$).

Let us see how the latter message is processed. Since there are no incoming hyperedges into term $a3$, n remains 0, and the processing of the message is concluded by the sending of the message TELL($q3$, $I(a3)$) to P_a.

3.2.4 TELL

When a peer receives a TELL(*QID*,R) message (see Figure 8), *QID* is an open call of some log object in

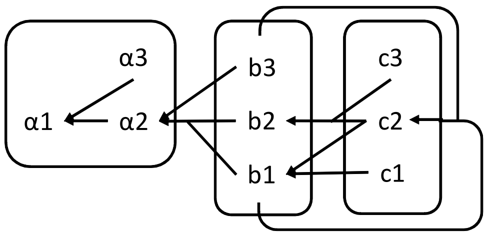

Figure 6: A network taxonomy B-graph

```
ASK(PID,QID: ID; $t$ : term; $A$ : set of terms);
 1.  $n \leftarrow 0$;  $QP, Q \leftarrow \emptyset$
 2.  for each hyperedge $h = \langle\{u_1, ..., u_r\}, t\rangle$ such that $\{u_1, ..., u_r\} \cap A = \emptyset$ do
 3.      $C \leftarrow \emptyset$
 4.      for each $u_i$ do
 5.          ID $\leftarrow$ NEW-QUERY-ID
 6.          $C \leftarrow C \cup \{ID\}$
 7.          $n \leftarrow n + 1$
 8.          ENQUEUE($Q, (P_h$, ID, $u_i$, $A \cup \{u_i\}))$
 9.      QP$\leftarrow$ QP$\cup \{C\}$
10.  if $n > 0$ then
11.      PERSIST(PID,QID,$t$,$n$,QP)
12.      until $Q \neq \emptyset$ do
13.          ($P_h$,ID,u,B) $\leftarrow$ DEQUEUE($Q$)
14.          $P_h$ : ASK($self$, $ID$, $u$, $B$)
15.  else PID:TELL(QID,$I(t)$)
```

Figure 7: The procedure to process ASK messages

the peer's log, in the program of some term (sub)query t with id *QID*$_1$. Then, as a first action, the peer retrieves this object by using the DELETE1 function, which takes as input *QID*, returns the object and *deletes* it form the log. Notice that there is exactly one object having *QID* as open call, since ASK generates a new id for each sub-query it identifies, as we have already seen. After retrieving the log object, TELL uses CLOSE to modify the query program *QP* in it, by closing the open call *QID*: this means to replace *QID* by R, obtaining a new query program *QP*$_1$. On line 3, the number of open calls of the log object is tested: if it is 1, then the just closed call was the last one to be open in query *QID*$_1$; in this case, the result of *QID*$_1$ is computed in S by COMPUTE-ANSWER. For a given program:

$$QP = \{SP_1, \dots, SP_m\}$$

where each sub-program SP_j is given by a collection of object sets:

$$SP_j = \{R_1^j, \dots, R_{m_j}^j\}$$

COMPUTE-ANSWER returns:

$$S = \bigcup_{j=1}^{m} \bigcap_{i=1}^{m_j} R_i^j$$

$S \cup I(t)$ is exactly what the QE procedure computes. If t is not in the terminology of the peer ($t \notin T_{self}$) then it follows that *QID*$_1$ is the id of the original query q. Thus, $I(t) = \emptyset$ and $S = ans(t, S_{\mathcal{N}})$. Therefore, the object (*PID*, *QID*$_1$, $t, 0, S$) is persisted in the log (line 5), indicating to QUERY(q) (Figure 5) that the evaluation of the query q has finished. Otherwise, the so obtained result $S \cup I(t)$ is TELL-ed to the peer *PID* which, according to the log object, was the one to ASK the evaluation of *QID*$_1$. Notice that this may fire another TELL message, in case *QID*$_1$ is the last open call of some other query. If the test on line 3 fails, then there are still open calls in the log object, which is therefore persisted back by PERSIST on line 6, after decreasing the number of open calls in it and replacing the query program *QP* by the updated one *QP*$_1$.

```
TELL(QID: ID; $R$ : set of objects);
1.  (PID, QID$_1$, $t$, $n$, QP) $\leftarrow$ DELETE1(QID)
2.  QP$_1$ $\leftarrow$ CLOSE(QP, QID, $R$)
3.  if $n = 1$ then
4.      $S \leftarrow$ COMPUTE-ANSWER(QP$_1$)
5.      if $t \notin T_{self}$ then PERSIST(PID, QID$_1$, $t, 0, S$)
6.      else PID:TELL(QID$_1$, $S \cup I(t)$)
7.  else PERSIST(PID, QID$_1$, $t$, $n-1$, QP$_1$)
```

Figure 8: The procedure to process TELL messages

In our example, the message TELL($q3$, $I(a3)$) is received by peer P_a. The function DELETE1 returns the log object (P_a, $q1$, t, 2, $\{\{q2, q3\}\}$), the only one that has the open call $q3$. CLOSE produces the new query program $\{\{q2, I(a3)\}\}$, and since n is not 1, the following modified log object is persisted:

$$(P_a, q1, t, 1, \{\{q2, I(a3)\}\}).$$

The example is completed in appendix.

3.3 Correctness and Complexity

As it has been argued, the combined action of the procedures processing ASK and TELL messages is equivalent to the behavior of the procedure QE. To see why in more detail, it suffices to consider the following facts:

1. An ASK message is generated for each recursive call performed by QE and vice-versa, that is whenever QE would perform a recursive call, an ASK message is generated. This is guaranteed by the fact that the test on line 2 of ASK is the same as the test on line 3 of QE. Therefore, the number of ASK messages is the same as the number of terms that can be found on a B-path from t.

2. For each ASK message, at most one log object is generated and persisted.

3. For each ASK message, a TELL message results, and no more. This can be observed by considering that, for each processed ASK message, there can be two cases:

   (a) no hyperedge is found that passes the test on line 2 of ASK: in this case, no subsequent ASK message is generated, and a TELL message is generated;

   (b) at least one hyperedge passes the test: in this case a number of sub-queries is generated and registered in the query program of the log object. Each such sub-query is evaluated by issuing an ASK message with a larger set of visited terms. Since the B-graph is finite, eventually each sub-query will lead to a term falling in the previous case (this is how QE terminates). When all sub-queries of a given term query t are closed, the number of open calls of t goes down to 0, and TELL issues another TELL message on t. This will propagate closure up, until all open calls are closed.

4. Finally, the COMPUTE-ANSWER procedure performs the same operation on the result of subqueries as QE does on the results of its recursive calls.

As a consequence of these facts we have the correctness of the network query evaluation procedure, and also its efficiency. In fact, the total number of messages generated is twice the number of terms visited by QE, and the number of log objects is no larger than that.

4 OPTIMIZATION ISSUES

So far, we have focused on correctness. In this Section we discuss optimization. A strong point of our model is that the adoption of caches could significantly speed up the evaluation of queries, by reducing both the latency time and the network throughput. This is because the set of queries that a peer can send to its articulated peers is bounded in size and can be pre-determined: it comprises all "foreign" queries of the peer, *i.e.* queries that appear as left-hand sides in the peer's articulations.

The subsequent subsections present three caching policies, namely:

- caching answers of local terms,
- caching answers of local terms and pushing answers of articulation tails, and
- caching answers of articulation heads.

4.1 Caching Answers of Local Terms

According to this caching policy, each peer $\mathcal{S}$ caches pairs of the form $(t, ans(t, S_{\mathcal{N}}))$, where t is a term in the peer's terminology $T_{\mathcal{S}}$. If there are no memory limitations for caches, then after a while each peer will have cached its whole terminology, and query evaluation reduces to locally calculating the extension of the query by union-ing and intersecting the extensions of the peer's terms. In other words, any peer will be able to evaluate network queries over its own taxonomy without sending any message to the network[1]! This is of course the idealistic case. In general, only some terms (possibly none) will be cached in each peer. Under these circumstances, when a peer $\mathcal{S}$ receives an ASK message for a term query t, the ASK procedure checks which of the answers for the term (sub)queries needed for the evaluation of t are in the cache, and issues ASK messages only for evaluating the remaining terms.

The modified query evaluation algorithms for supporting this caching policy are parts of the algorithms for the more general policy that is described in Section 4.2.

[1] Apart those required for re-evaluating queries when updates occur.

4.2 Caching Answers of Local Terms and Pushing Answers of Articulation Tails

A complementary scenario, best suited for a P2P system that offers recommendation services in push-style manner, is to assume that each peer $\mathcal{S}$ knows also the articulations $t_1 \wedge \ldots \wedge t_r \preceq u$ from other peers $\mathcal{S}'$ to $\mathcal{S}$ (called *foreign articulations*). In this case, if all the terms $t_1, \ldots, t_r$ are cached in $\mathcal{S}$, then $\mathcal{S}$ can send to $\mathcal{S}'$ the pair $(t_1 \wedge \ldots \wedge t_r, ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}}))$ to be stored in the cache of $\mathcal{S}'$. This can be done because from Definition 4 it follows that

$$ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}}) = \bigcap \{ans(t_i, S_{\mathcal{N}}) \mid 1 \leq i \leq r\}$$

The cache is exploited by the modified ASK procedure (ASK$_c$), shown in Figure 9. The modified with caching TELL procedure (TELL$_c$) is shown in Figure 10. The modifications are indicated by bold line numbers and are described in a semi-formal way, in order to abstract from irrelevant details.

The cache of a peer $\mathcal{S}$ consists of two kinds of pairs:

- $(t', ans(t', S_{\mathcal{N}}))$ where t' is a term in the peer's terminology $T_{\mathcal{S}}$. Pairs of this kind are inserted into the cache by the TELL$_c(QID, t', R)$ procedure[2], when the peer $\mathcal{S}$ is TELL-ed the answer R for a term query t', initiated by an ASK message of type

  ASK(*PID*, *QID*, t', $\{u, t'\}$)

  where u is a new term created by QUERY(q) to represent the original (complex) query q, posed to peer $\mathcal{S}$. This means that the term t' appears in q and is not evaluated in the context of the evaluation of a more general term. For example, this is the case of the ASK messages presented at the end of Section 3.2.3:

  1. $(P_a$, $q2$, $a2$, $\{t, a2\})$, and
  2. $(P_a$, $q3$, $a3$, $\{t, a3\})$.

  In this way, based on the correctness of the QUERY procedure (Section 3.3), it is guaranteed that $R = ans(t', S_{\mathcal{N}})$, *i.e.* the received answer R is the full answer for t' and not a subset of it, reduced due to cycles in the taxonomy $(T_{\mathcal{N}}, \preceq_{\mathcal{N}})$. Thus, the pair (t', R) can be safely cached.

[2] Note that TELL$_c(QID, t', R)$ takes an extra argument t', which is the term query corresponding to query id QID.

```
ASK_c(PID,QID: ID; $t$ : term; $A$ : set of terms);
1.  if $t$ is cached then PID:TELL_c(QID, $t$, $ans(t, S_{\mathcal{N}})$)
2.  else if $|A| = 2$ then add $t$ into TO-BE-CACHED log // $t$ is a term of the original query $q$
3.      $n \leftarrow 0$;  $QP, Q, S \leftarrow \emptyset$
4.      for each hyperedge $h = \langle\{u_1, ..., u_r\}, t\rangle$ such that $\{u_1, ..., u_r\} \cap A = \emptyset$ do
5.          if $P_h \neq self$ and $u_1 \wedge \ldots \wedge u_r$ is cached then $C \leftarrow \{ans(u_1 \wedge \ldots \wedge u_r, S_{\mathcal{N}})\}$
6.          else $C \leftarrow \emptyset$
7.              for each $u_i$ do
8.                  if $P_h = self$ and $u_i$ is cached then
9.                      $C \leftarrow C \cup \{ans(u_i, S_{\mathcal{N}})\}$
10.                 else
11.                     ID $\leftarrow$ NEW-QUERY-ID
12.                     $C \leftarrow C \cup \{ID\}$
13.                     $n \leftarrow n + 1$
14.                     ENQUEUE($Q$, ($P_h$, ID, $u_i$, $A \cup \{u_i\}$))
15.         QP$\leftarrow$ QP$\cup \{C\}$
16.     if $n > 0$ then
17.         PERSIST(PID,QID,$t$,$n$,QP)
18.         until $Q \neq \emptyset$
19.             ($P_h$,ID,$u$,$B$) $\leftarrow$ DEQUEUE($Q$)
20.             $P_h$:ASK_c($self$, $ID$, $u$, $B$)
21.     else if QP$\neq \emptyset$ then $S \leftarrow$ COMPUTE-ANSWER(QP)
22.         PID:TELL_c(QID, $t$, $S \cup I(t)$)
```

Figure 9: The procedure to process ASK messages with cache

```
TELL_c(QID: ID; $t'$ : term; $R$ : set of objects);
1.  if $t'$ in TO-BE-CACHED log then // $t'$ is a term of the original query $q$
2.      delete $t'$ from TO-BE-CACHED log
3.      CACHE($t'$, $R$)
4.      for each foreign articulation $t_1 \wedge \ldots \wedge t_r \preceq u$ from another peer $\mathcal{S}$ to $self$ do
5.          if $t' \in \{t_1, ..., t_r\}$ and all $t_1$, ..., $t_r$ are cached then
6.              forward to $\mathcal{S}$ the pair $(t_1 \wedge \ldots \wedge t_r, ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}})$ for caching
7.  (PID, $QID_1$, $t$, $n$, QP) $\leftarrow$ DELETE1(QID)
8.  $QP_1$$\leftarrow$ CLOSE(QP, QID, $R$)
9.  if $n = 1$ then
10.     $S \leftarrow$ COMPUTE-ANSWER($QP_1$)
11.     if $t \notin T_{self}$ then PERSIST(PID, $QID_1$, $t$, 0, $S$)
12.     else PID:TELL_c($QID_1$, $t$, $S \cup I(t)$)
13. else PERSIST(PID, $QID_1$, $t$, $n-1$, $QP_1$)
```

Figure 10: The procedure to process TELL messages with cache

- $(t_1 \wedge \ldots \wedge t_r, ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}}))$ where $t_1 \wedge \ldots \wedge t_r \preceq u$ is an articulation from $\mathcal{S}$ to $\mathcal{S}'$, *i.e.* $u \in T_{\mathcal{S}}$ and $t_1, \ldots, t_r \in T_{\mathcal{S}'}$. Each such pair is forwarded to $\mathcal{S}$ by the TELL$_c$ procedure executed at the peer $\mathcal{S}'$, upon realizing that all the terms involved in the left-hand side of the articulation are stored in the local (to $\mathcal{S}'$) cache. In particular, this check is made immediately after a pair $(t', ans(t', S_{\mathcal{N}}))$ is added in the cache of $\mathcal{S}'$, where $t' \in \{t_1, ..., t_r\}$ (see lines 3-6 of TELL$_c$).

Below are the main differences of ASK$_c$(*PID*, *QID*, t, A) with respect to the cache-less ASK:

- If the answer to the term query t ASK-ed by peer PID is in the cache, then the answer is immediately TELL-ed to peer PID. Otherwise, if $|A| = 2$ then t is added in the TO-BE-CACHED log (t is a term of the original query q). The TO-BE-CACHED log is checked by TELL$_c(QID, t', R)$. If t' is found in the TO-BE-CACHED log then (t', R) is added to the local cache through the CACHE(t', R) command (line 3 of TELL$_c$).

- Before processing the tail of a hyperedge h which passes the test on line 4, a test is performed, to ascertain whether the query corresponding to the tail, given by $u_1 \wedge \ldots \wedge u_r$, is in the cache (this test is needed only if $P_h \neq self$, *i.e.* h corresponds to an articulation hyperedge). If yes, the only action taken is the insertion of $ans(u_1 \wedge \ldots \wedge u_r, S_{\mathcal{N}})$ into the query sub-program *QP* being built (line 15). If the query is not in the cache, then for each u_i, it is checked if its answer is in the cache (this test is needed only if $P_h = self$). If not, then the execution proceeds normally.

- If all sub-queries are cached, then when all relevant hyperedges have been processed (line 16), n is zero but *QP* is not empty. In this case the test on line 21 is passed, and the result of *QID* is computed in S as if closing *QP* in a TELL. S is subsequently returned along with $I(t)$. If *QP* is empty, then no hyperedge has been found and $S = \emptyset$. So, the result returned to the user is simply $I(t)$.

We would like to note that our algorithms can further be extended such that TELL$_c$ caches the answer $S \cup I(t)$ for term sub-queries t before TELL-ing them to the requesting peer PID (line 12 of TELL$_c$), as long as it is certain that $S \cup I(t) = ans(t, S_{\mathcal{N}})$. This is the case if (i) for each term u of a peer $\mathcal{S}'$ encountered during the evaluation of t (including t itself), all hyperedges $\langle \{u_1, ..., u_r\}, u \rangle$ of the taxonomy B-graph of $\mathcal{S}'$ pass the test of line 4 of ASK$_c$, or (ii) u is cached. Thus, (i) no evaluation path of u is eliminated due to cycles in the taxonomy $(T_{\mathcal{N}}, \preceq_{\mathcal{N}})$ or (ii) $ans(u, S_{\mathcal{N}})$ is immediately retrieved from the cache.

For this reason PERSIST(*PID*,*QID*,t, n,*QP*) and TELL$_c$(*QID*,t', R) should be extended with an extra field $flag$ that takes the values `full` or `partial`. A (query) log object (*PID*,*QID*,t, n, *QP*, $flag$), where $flag =$`full`, of a peer $\mathcal{S}$ indicates that (i) for all *closed* term sub-queries of QP, full answers have been received and (ii) all hyperedges $\langle \{u_1, ..., u_r\}, t \rangle$ of the taxonomy B-graph of $\mathcal{S}$ have passed the test of line 4 of ASK$_c$. If this is not the case, $flag =$`partial`. A message TELL$_c(QID, t', R, flag)$, where $flag =$`full`, indicates that $R = ans(t', S_{\mathcal{N}})$, whereas a message TELL$_c(QID, t', R, flag)$, where $flag =$`partial`, indicates that $R \subseteq ans(t', S_{\mathcal{N}})$. Thus, based on the $flag$ information, the TELL$_c$ procedure executed at a peer will always be able to know if the computed answer $S \cup I(t)$ for a term sub-query t requested by peer PID is a full or partial answer. In the case of a full answer and if t is the head of an articulation hyperedge then $(t, S \cup I(t))$ is cached. We want to note that the latter condition is not a strong condition and is needed only in order to reduce the cache size, while taking the most advantage of caching.

The extended ASK$_c$ procedure (ASK$_c^{ext}$) and the extended TELL$_c$ procedure (TELL$_c^{ext}$) are given in Figures 11 and 12, respectively. The modifications are indicated by bold line numbers. Note that TELL$_c^{ext}$ calls the procedure CACHE&FORWARD (Figure 13), when a pair $(t, ans(t, S_{\mathcal{N}}))$ is going to be stored in the cache. Additionally, TELL$_c^{ext}$ uses the function $min(flag, flag')$ (lines 8, 11), which returns the minimum of the flag values $flag$, $flag'$, based on the ordering `partial` $\leq$ `full`. This guarantees that the flag value of the TELL$_c^{ext}$ message in line 8 and the log object in line 11 is correct.

4.3 Caching Answers of Articulation Heads

The previous algorithms will cache the most frequently used terms, taking full advantage of caching with no extra cost for computing cached answers. However, caches may get filled very quickly. Below we investigate the case that we cache only the heads of articulation hyperedges, as the cached answer of these terms is the most beneficial for speeding-up query answering. For instance, in the example of Figure 6, we want to cache only a_2 on Peer P_a, b_1 and b_2 on Peer P_b, and c_2 on Peer P_c.

For this alternative caching case, a top algorithm can be easily designed such that whenever a peer receives an external query q, it finds the local terms that are heads of articulation hyperedges and are needed for the

```
ASK$_c^{ext}$(PID,QID: ID; $t$ : term; $A$ : set of terms);
1.  if $t$ is cached then PID:TELL$_c^{ext}$(QID, $t$, $ans(t, S_{\mathcal{N}})$, full)
2.  else if $|A| = 2$ then add $t$ into TO-BE-CACHED log // $t$ is a term of the original query $q$
3.      $n \leftarrow 0$;  QP, $Q$, $S \leftarrow \emptyset$; $flag$ =full
4.      for each hyperedge $h = \langle\{u_1, ..., u_r\}, t\rangle$ do
5.          if $\{u_1, ..., u_r\} \cap A = \emptyset$ then
6.              if $P_h \neq self$ and $u_1 \wedge \ldots \wedge u_r$ is cached then $C \leftarrow \{ans(u_1 \wedge \ldots \wedge u_r, S_{\mathcal{N}})\}$
7.              else $C \leftarrow \emptyset$
8.                  for each $u_i$ do
9.                      if $P_h = self$ and $u_i$ is cached then
10.                         $C \leftarrow C \cup \{ans(u_i, S_{\mathcal{N}})\}$
11.                     else
12.                         ID $\leftarrow$ NEW-QUERY-ID
13.                         $C \leftarrow C \cup \{ID\}$
14.                         $n \leftarrow n + 1$
15.                         ENQUEUE($Q$, ($P_h$, ID, $u_i$, $A \cup \{u_i\}$))
16.             QP$\leftarrow$ QP$\cup$ $\{C\}$
17.         else $flag$ =partial
18.     if $n > 0$ then
19.         PERSIST(PID, QID, $t$, $n$, QP, $flag$)
20.         until $Q \neq \emptyset$
21.             ($P_h$,ID,u,B) $\leftarrow$ DEQUEUE($Q$)
22.             $P_h$:ASK$_c^{ext}$($self$, ID, $u$, $B$)
23.     else if QP$\neq \emptyset$ then $S \leftarrow$ COMPUTE-ANSWER(QP)
24.         PID:TELL$_c^{ext}$(QID, $t$, $S \cup I(t)$, $flag$)
```

Figure 11: The extended procedure to process ASK messages with cache

```
TELL$_c^{ext}$(QID: ID; $t'$ : term; $R$ : set of objects; $flag'$ : {full, partial});
1.  if $t'$ in TO-BE-CACHED log then // $t'$ is a term of the original query $q$
2.      CACHE&FORWARD($t'$, $R$)
3.  (PID, QID$_1$, $t$, $n$, QP, $flag$) $\leftarrow$ DELETE1(QID)
4.  QP$_1$$\leftarrow$ CLOSE(QP, QID, R)
5.  if $n = 1$ then
6.      $S \leftarrow$ COMPUTE-ANSWER(QP$_1$)
7.      if $t \notin T_{self}$ then PERSIST(PID, QID$_1$, $t$, 0, $S$, full)
8.      else PID:TELL$_c^{ext}$(QID$_1$, $t$, $S \cup I(t)$, $min(flag, flag')$)
9.          if $min(flag, flag')$=full and $t$ is the head of an articulation hyperedge then
10.             CACHE&FORWARD($t$,$S \cup I(t)$)
11. else PERSIST(PID, QID$_1$, $t$, $n - 1$, QP$_1$, $min(flag, flag')$)
```

Figure 12: The extended procedure to process TELL messages with cache

```
CACHE&FORWARD($t$ : term; $R$ : set of objects);
  // It stores the pair $(t, R)$ in the local cache and checks if related (foreign articulation)
     query-answer pairs can be forwarded to other peers for caching
1.      CACHE($t, R$)
2.      if $t$ in TO-BE-CACHED log then delete $t$ from TO-BE-CACHED
3.      for each foreign articulation $t_1 \wedge \ldots \wedge t_r \preceq u$ from another peer $\mathcal{S}$ to $self$ do
4.          if $t \in \{t_1, ..., t_r\}$ and all $t_1$, ..., $t_r$ are cached then
5.              forward to $\mathcal{S}$ the pair $(t_1 \wedge \ldots \wedge t_r, ans(t_1 \wedge \ldots \wedge t_r, S_{\mathcal{N}})$ for caching
```

Figure 13: The procedure CACHE&FORWARD

```
$\textsc{Ask}_c^{alt}$(PID,QID: **ID**; $t$ : **term**; $A$ : **set of terms**);
1.  **if** $t$ is cached **then** PID:$\textsc{Tell}$(QID, $t$, $ans(t, S_{\mathcal{N}})$)
2.  **else** $n \leftarrow 0$; QP, $Q$, $S \leftarrow \emptyset$
3.      **for each** hyperedge $h = \langle\{u_1, ..., u_r\}, t\rangle$ **such that** $\{u_1, ..., u_r\} \cap A = \emptyset$ **do**
4.          $C \leftarrow \emptyset$
5.          **for each** $u_i$ **do**
6.              **if** $u_i$ is cached **then**
7.                  $C \leftarrow C \cup \{ans(u_i, S_{\mathcal{N}})\}$
8.                  **else**
9.                      ID $\leftarrow$ $\textsc{New-query-id}$
10.                 $C \leftarrow C \cup \{ID\}$
11.                 $n \leftarrow n + 1$
12.                 $\textsc{Enqueue}(Q, (P_h, ID, u_i, A \cup \{u_i\}))$
13.         QP$\leftarrow$ QP$\cup \{C\}$
14.     **if** $n > 0$ **then**
15.         $\textsc{Persist}$(PID,QID,$t$,$n$,QP)
16.         **until** $Q \neq \emptyset$
17.             ($P_h$,ID,u,B) $\leftarrow$ $\textsc{Dequeue}(Q)$
18.             $P_h$:$\textsc{Ask}_c^{alt}(self, ID, u, B)$
19.     **else if** QP$\neq \emptyset$ **then** $S \leftarrow$ $\textsc{Compute-answer}$(QP)
20.         PID:$\textsc{Tell}$(QID, $S \cup I(t)$)
```

Figure 14: An alternative procedure to process ASK messages with cache

evaluation of the query. Then, for each such term t, if t is not cached, it calls the QUERY(t) procedure (Figure 5) and it caches t along with the received answer R, as it is certain that $R = ans(t, S_{\mathcal{N}})$. This will fill the needed caches. The answer of the original query is then computed locally. Note that QUERY(t), in this case, should call $\textsc{Ask}_c^{alt}$ (Figure 14) which is a simplified version of $\textsc{Ask}_c$ that issues $\textsc{Ask}_c^{alt}$ and TELL messages. Though this approach has the extra cost of requiring full answers for terms that do not belong to the original query q, it is the most beneficial with respect to the trade-off cache size versus speed.

Of course, another alternative is if the above mentioned top algorithm asks for the answers of foreign terms t (through QUERY(t)) that appear in the body of articulation hyperedges, instead of asking for the answers of (local) terms t that are heads of articulation hyperedges.

4.4 Synopsis

Above we described three caching policies. Overall, four query evaluation modes can be supported by our model. The three caching policies result in faster query evaluation, but possibly not very updated results, since taxonomies, interpretations and articulations change. The mode without cache results in fresher results but with a slower query evaluation.

In case there are memory limitations for caches, various update policies could be employed, *e.g.* keep in cache only the answers of the most frequently used terms, or keep in cache only some parts of the answers, for instance "popular" objects according to some external information collected for this purpose (object-ranking techniques similar to page-ranking techniques for the Web could be employed to this end).

5 Related work

In this paper we studied the problem of evaluating content-based retrieval queries in an entirely pure P2P architecture (without any form of structuring), where each peer can have its own conceptual model expressed as a taxonomy.

Note that the peers of our model are quite autonomous in the sense that they do not have to share or publish their stored objects, taxonomies or mappings with the rest of the peers (neither to one central server, nor to the on-line peers). To participate in the network, a peer just has to answer the incoming queries by using its local base, and to propagate queries to those peers that according to its "knowledge" (i.e. taxonomy + articulations) may contribute to the evaluation of the query. However both of the above tasks are optional and at the "will" of the peer.

There have been several works on P2P systems endowed with logic-based models of the peers' information bases and of the mappings relating them (called *P2P mappings*). These works can be classified in two broad categories: (1) those assuming propositional or Horn clauses as representation language or as a computational framework, and (2) those based on more powerful formalisms. With respect to the former category (*e.g.*, see [3, 4, 2]), our work makes an important contribution, by providing a much simpler algorithm for performing query answering than those based on resolution. Indeed, we do rely on the theory of propositional Horn clauses, but only for proving the correctness of our algorithm. For implementing query evaluation, we devise an algorithm that avoids the (unnecessary) algorithmic complications that plague the methods based on resolution. As an example, after appropriate transformations our framework can be seen as a special case of that in [4]. Then, query evaluation can be performed by first computing the prime implicates of the negation of each term in the query, using the resolution-based algorithms presented in [4]. As the complexity of this problem is exponential w.r.t the size of the taxonomy and polynomial w.r.t. the size of Obj, there is no computational gain in using this approach. Instead, there is an algorithmic loss, since the method is much more complicated than ours. Additionally, for each (sub)query posed to a peer, its prime implicates are returned back to the peer one by one (though asynchronous messaging). In our case its answer is returned back to the peer that posed the query (also through asynchronous messaging) in single message.

As for the second category above, works in this area have focused on providing highly expressive knowledge representation languages in order to capture at once the widest range of applications. Notably, [10] proposes a model allowing, among other things, for existential quantification both in the bodies and in the heads of the mapping rules. Inevitably, such languages pose computational problems: deciding membership of a tuple in the answer of a query is undecidable in the framework proposed by [10], while disjunction in the rules' heads makes the same problem coNP-hard already for datalog with unary predicate (*i.e.* terms) [27]. These problems are circumvented by changing the semantics of a P2P network, in particular by adopting an epistemic reading of mappings. As a result, the inferential relation of the resulting logic is weakened up to the point of making the above mentioned decision problem solvable in polynomial time.

Below, we review in more detail several works dealing with the problem of answering (union of) conjunctive queries posed to a peer in logic-based P2P frameworks.

In [8], a query answering algorithm for simple P2P systems is presented where each peer $\mathcal{S}$ is associated with a local database, an (exported) peer schema, and a set of local mapping rules from the schema of the local database to the peer schema. P2P mapping rules are of the form $cq_1 \rightsquigarrow cq_2$, where cq_1, cq_2 are conjunctive queries of the same arity $n \geq 1$ (possibly involving existential variables), expressed over the union of the schemas of the peers, and over the schema of a single peer, respectively[3]. Note that this representation framework partially subsumes our network source framework, since in our case cq_1, cq_2 are of arity 1, cq_1 is a conjunctive query of the form $u_1(x) \wedge ... \wedge u_r(x)$ over the terminology of a single peer[4] and cq_2 is a single atom query $t(x)$ over the terminology of the peer that the mapping (articulation) belongs to. However, simple P2P systems cannot express the local to a peer $\mathcal{S}$ taxonomy $\preceq_{\mathcal{S}}$ of our framework. Query answering in simple P2P systems according to the first-order logic (FOL) semantics is in general undecidable. Therefore, the authors adopt a new semantics based on epistemic logic in order to get decidability for query answering. Notably, the FOL semantics and epistemic logic semantics for our framework coincide. In particular, in [8], a centralized bottom-up algorithm is presented which essentially constructs a finite database RDB which constitutes a "representative" of all the epistemic models of the P2P system. The answers to a conjunctive query q are the answers of q w.r.t. RDB. However, though this algorithm has polynomial time complexity, it is centralized. In contrast, we present distributed algorithms based on asynchronous messaging.

The work in [10] extends the work in [8] where the same framework and epistemic semantics are considered. A top-down distributed query answering algorithm is presented which is based on synchronous messaging. Essentially, the algorithm returns to the peer where the (sub)query is posed, a datalog program containing the full extensions of the relevant to the query, peer source predicates. The returned to the user datalog program is used for providing the answers to the user query. Though the algorithm for a user query and a term subquery visits the corresponding peer just once, it is based on synchronous messaging with all the subsequent delays. Note that our algorithm is based on asynchronous messaging.

The framework in [30], extends our framework by considering (i) n-ary (instead of unary) predicates (*i.e.* P2P mappings are general datalog rules) and (ii) a set of domain relations (also suggested in [31, 29]), mapping the objects of one peer to the objects of another peer. A distributed query answering algorithm is presented based on synchronous messaging. However, the algorithm will perform poorly in our restricted framework[5], since when a peer receives a (sub)query, it

[3] Note that P2P mapping rules of this kind can accommodate both GAV and LAV-style mappings, and are referred in the literature as GLAV mappings.

[4] Recall that this restriction can be easily relaxed.

[5] In our framework, domain relations correspond to the identity relation.

iterates through the relevant P2P mappings and for each one of them, sends a (sub)query to the appropriate peer (waiting for its answer), until fixpoint is reached. In our case, when a peer receives a (sub)query, each relevant P2P mapping is considered just once and no iteration until fixpoint is required. Additionally, asynchronous messaging is supported.

A P2P framework similar to [8] is presented in [24], where query answering according to FOL semantics is investigated. Since in general, query answering is undecidable, the authors present a centralized algorithm (employed in the Piazza system [22]), which however is complete (the algorithm is always sound), only for the case that polynomial time complexity in query answering can be achieved. This includes the condition that inclusion P2P mappings are acyclic. However, such a condition severely restricts the modularity of the system. Note that our algorithm is sound and complete even in the case that there are cycles in the term dependency path and it always terminates. Thus, our framework allows placing articulations between peers without further checks. This is quite important, because the actual interconnections are not under the control of any actor in the system.

In [19, 18], the authors consider a framework where each peer is associated with a relational database, and P2P mapping rules contain conjunctive queries in both the head and the body of the rule (possibly with existential variables), each expressed over the alphabet of a single peer. Again the semantics of the system is defined based on epistemic logic [17]. In these papers, a peer database update algorithm is provided allowing for subsequent peer queries to be answered locally without fetching data from other nodes at query time. The algorithm (which is based on asynchronous messaging) starts at the peer which sends queries to all neighbor peers according to the involved mapping rules. When a peer receives a query, the query is processed locally by the peer itself using its own data. This first answer is immediately replied back to the node which issued the query and sub-queries are propagated similarly to all neighbor peers. When a peer receives an answer, (i) it stores the answer locally, (ii) it materializes the view represented in the head on the involved mapping rule, and (ii) it propagates the result to the peer that issued the (sub)query. Answer propagation stops when no new answer tuples are coming to the peer through any dependency path, that is until fixpoint is reached. In our case, the database update problem for a peer $\mathcal{S}$ amounts to invoking $\mathcal{S}'$: $\text{QUERY}(q)$ for each articulation $q \preceq t$ from $\mathcal{S}$ to another peer $\mathcal{S}'$ and storing the answer locally to $\mathcal{S}$. Note that our query answering algorithm is also based on asynchronous messaging. However, since it considers a limited framework, it is much simpler and no computation until fixpoint is required. In particular, for each term (sub)query issued to a peer through ASK, only one answer is returned through TELL.

The authors in [29] introduce a model of a peer database management system (PDMS) which uses a mapping that combines schema-level and data-level mappings. They call this new type of mapping *bi-level mapping*. The authors verbally describe a query evaluation procedure for the PDMS that uses the bi-level mappings but not specific algorithms are provided. Additionally, the presence of cycles is not addressed. In our case, we consider P2P taxonomy-based sources and distributed algorithms are provided which work even in the presence of cycles.

In [9], extending [10], an epistemic semantics has been proposed in order to deal with possible inconsistencies in P2P Data Inference Systems formalized in a first order multi-modal language. The authors consider the case of local inconsistency, as well as global inconsistency. Mappings are formalized in such a way that they cannot be used to propagate information from some locally inconsistent theory. Moreover mappings can only be used to propagate information to a peer, as far as they do not contradict either local information or other non-local information that may be deduced on that peer. The authors present distributed algorithms, based on synchronous messaging, which extend these of [10] in such a way that they can handle conflicts. Comments similar to these that compare our algorithm with [10] apply here.

The authors in [7] address the problem of P2P query answering over distributed propositional information sources that may be mutually inconsistent. They assume the existence of a priority ordering over the peers to discriminate between peers with conflicting information. This ordering may reflect an individual's level of trust in an information source. They provide for their framework a distributed entailment relation related to argumentation frameworks. Decentralized algorithms for computing answers YES, NO, UNKNOWN to arbitrary CNF queries according to distributed entailment are presented. The algorithms, which are based on the algorithms of [4] are very complicated and dedicated to the inconsistencies that may arise. In our case, we proposed much more simpler algorithms for P2P taxonomy-based information sources, where the answer to a query is a set of objects.

The work in [11] investigates the data exchange problem among distributed consistent P2P deductive databases with integrity constraints and uses dynamic preferences to drive the integration process in the case of conflicting information. The interaction among peer deductive databases has been modeled by importing, by means of mapping rules, maximal sets of atoms not violating integrity constraints, that is maximal sets of atoms that allow the peer to enrich its knowledge while preventing inconsistencies. The proposed semantics, called *maximal weak model semantics*, can be computed by means of a prioritized logic program. However, no distributed algorithms are provided.

The authors in [12] propose a logic-based framework for modeling the interaction among possibly inconsistent P2P deductive databases. Data are exchanged by means of mapping rules. Under this semantics, called *minimal weak model semantics*, each peer just imports the minimal information allowing to restore consistency. A prioritized logic program is defined which computes the minimal weak model semantics. However, no distributed algorithms are provided. In [14], the same framework and semantics are considered but these are computed through a centralized disjunctive logic programs.

The work in [13], extends the works in [11, 12] by considering two types of of mapping rules. One that imports a maximal set of atoms as long as the peer remains consistent and the other that imports a minimal set of atoms as long as they imply the satisfaction of some peer integrity constraints. The semantics, called *preferred weak model semantics*, are computed through a centralized prioritized logic program with two levels of priorities. As in [11, 12], no distributed algorithms are provided.

In [6], the authors develop a P2P distributed algorithm for query evaluation in a multi-context system framework that is based on propositional defeasible logic. P2P mappings are defeasible rules and each peer holds strict rules, defeasible rules, and a total preference ordering of peers for resolving conflicts when conflicting information is imported. Each query is a literal l issued to some peer and the answer is that l is a provably (not) logical conclusion of the system or this remain open. In our case, we proposed much more simpler algorithms for P2P taxonomy-based information sources, where the answer to a query is a set of objects.

In [16], the authors present P2P algorithms for model building in heterogeneous non-monotonic multi-context systems where P2P mappings are non-monotonic rules. In our case, we are not interesting in model building but in query answering.

With respect to [9, 7, 11, 12, 14, 13, 6], we can say that in our framework no conflicts arise. Additionally, note that in all works mentioned in the related work section, no optimizations based on caching are considered.

6 Conclusions

This study presents a model of a P2P network consisting of sources based on taxonomies. A taxonomy states subsumption relationships between negation-free DNF formulas on terms and negation-free conjunctions of terms. The language for querying such sources offers Boolean combinations of terms, in which negation can be efficiently handled by adopting a closed-world reading of the information. An efficient, hypergraph-based query evaluation method is presented for such sources, resting on results coming from the theory of propositional clauses. It is also shown that extending the expressive power of the taxonomy language by adding negation or full disjunction, leads to the intractability of the decision problem.

A model of a P2P network, having sources as nodes, is subsequently presented. The essential feature of the model is the possibility of relating the assumed disjoint peer terminologies by means of subsumption relationships of the same type as those in the taxonomies of the sources. The resulting system subscribes to the universally accepted notion of P2P information system, recently postulated also in the context of the so-called emergent semantics [1].

An efficient query evaluation procedure for queries stated against such a network is presented, and proved correct. The procedure is a distributed version of the centralized procedure, based on an asynchronous, message-based interaction amongst the peers aimed at favoring scalability. Some optimization techniques are also discussed, namely one based on caching, for which the algorithms for message processing are given.

Finally, the work is related to the most relevant papers in the area of P2P systems.

It is true that our P2P caching algorithms, as they have been described, work only for static datasets. Dynamic datasets can be handled by setting an expiration time for the cached terms. In particular, terms (along their answer) are removed from the cache when the time stored in the cache exceeds the set expiration time. Details will be given in future work. Future work also concerns implementation and evaluation of our framework.

References

[1] K. Aberer, T. Catarci, P. Cudré-Mauroux, T. S. Dillon, S. Grimm, M. Hacid, A. Illarramendi, M. Jarrar, V. Kashyap, M. Mecella, E. Mena, E. J. Neuhold, A. M. Ouksel, T. Risse, M. Scannapieco, F. Saltor, L. D. Santis, S. Spaccapietra, S. Staab, R. Studer, and O. D. Troyer, “Emergent Semantics Systems,” in *1st Intern. IFIP Conference on Semantics of a Networked World (ICSNW-2004)*, 2004, pp. 14–43.

[2] P. Adjiman, “Peer-to-peer reasoning in propositional logic: algorithms, scalability, study

and applications," Ph.D. dissertation, University of Paris South, 2006.

[3] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon, "Distributed reasoning in a peer-to-peer setting," in *Proceedings of ECAI 2004, the European Conference on Artificial Intelligence*, R. L. de Mntaras and L. Saitta, Eds., Valencia, Spain, 2004, pp. 945–946, short paper.

[4] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon, "Distributed reasoning in a peer-to-peer setting: Application to the semantic web," *Journal of Artificial Intelligence Research*, vol. 25, pp. 269–314, 2006.

[5] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu, "Data Management for Peer-to-Peer Computing: A Vision," in *Fifth International Workshop on the Web and Databases (WebDB-2002)*, Madison, Wisconsin, June 2002.

[6] A. Bikakis, G. Antoniou, and P. Hassapis, "Strategies for contextual reasoning with conflicts in ambient intelligence," *Knowledge and Information Systems (KAIS)*, vol. 27, no. 1, pp. 45–84, 2011.

[7] A. Binas and S. A. McIlraith, "Peer-to-Peer Query Answering with Inconsistent Knowledge," in *Eleventh International Conference on the Principles of Knowledge Representation and Reasoning (KR-2008)*, 2008, pp. 329–339.

[8] D. Calvanese, E. Damaggio, G. De Giacomo, M. Lenzerini, and R. Rosati, "Semantic Data Integration in P2P Systems," in *First International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P-2003)*, 2003, pp. 79–90.

[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Inconsistency tolerance in P2P data integration: An epistemic logic approach," *Information Systems*, vol. 33, no. 4-5, pp. 360–384, 2008.

[10] D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati, "Logical foundations of peer-to-peer data integration," in *23rd ACM symposium on Principles of database systems (PODS-2004)*. New York, NY, USA: ACM Press, 2004, pp. 241–251.

[11] L. Caroprese and E. Zumpano, "Handling Preferences in P2P Systems," in *7th International Symposium on Foundations of Information and Knowledge Systems (FoIKS-2012)*, 2012, pp. 91–106.

[12] L. Caroprese and E. Zumpano, "Restoring Consistency in P2P Deductive Databases," in *Scalable Uncertainty Management - 6th International Conference on Scalable Uncertainty Management (SUM-2012)*, 2012, pp. 168–179.

[13] L. Caroprese and E. Zumpano, "A Logic Based Approach for Managing Incompleteness and Inconsistencies in P2P Deductive Databases," in *19th International Symposium on Database Engineering & Applications*, 2015, pp. 168–173.

[14] L. Caroprese and E. Zumpano, "A Logic Based Approach for Restoring Consistency in P2P Deductive Databases," in *26th International Conference on Database and Expert Systems Applications (DEXA-2015)*, 2015, pp. 3–12.

[15] R. Cyganiak, D. Wood, and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax," W3C Recommendation, 25 February 2014, available at https://www.w3.org/TR/rdf11-concepts/.

[16] M. Dao-Tran, T. Eiter, M. Fink, and T. Krennwallner, "Distributed Evaluation of Nonmonotonic Multi-context Systems," *Journal of Artificial Intelligence Research (JAIR)*, vol. 52, pp. 543–600, 2015.

[17] E. Franconi, G. M. Kuper, A. Lopatenko, and L. Serafini, "A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems," in *First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P-2003)*, 2003, pp. 64–76.

[18] E. Franconi, G. M. Kuper, A. Lopatenko, and I. Zaihrayeu, "A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks," in *EDBT'04 Intern. Workshop on Peer-to-Peer Computing and Databases (P2P&DB-2004)*, 2004, pp. 446–455.

[19] E. Franconi, G. M. Kuper, A. Lopatenko, and I. Zaihrayeu, "Queries and Updates in the coDB Peer to Peer Database System," in *30th International Conference on Very Large Data Bases (VLDB-2004)*, 2004, pp. 1277–1280.

[20] G. Gallo, G. Longo, and S. Pallottino, "Directed Hypergraphs and Applications," *Discrete Applied Mathematics*, vol. 42, no. 2, pp. 177–201, 1993.

[21] B. Ganter and R. Wille, *"Formal Concept Analysis: Mathematical Foundations"*. Springer-Verlag, Heidelberg, 1999.

[22] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov, "The Piazza Peer Data Management System," *IEEE Transactions on*

Knowledge and Data Engineering, vol. 16, no. 7, pp. 787–798, 2004.

[23] A. Halevy, Z. Ives, P. Mork, and I. Tatarinov, "Piazza: Data Management Infrastructure for Semantic Web Applications," in *12th International Conference on World Wide Web (WWW-2003)*, May 2003, pp. 556 – 567.

[24] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov, "Schema Mediation in Peer Data Management Systems," in *19th International Conference on Data Engineering (ICDE-2003)*, March 2003, pp. 505–518.

[25] P. Hayes and P. F. Patel-Schneider, "RDF 1.1 Semantics," W3C Recommendation, 25 February 2014, available at http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.

[26] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space (1st edition)*. Morgan & Claypool, 2011.

[27] C. Meghini and Y. Tzitzikas, "Querying Articulated Sources," in *3rd Intern. Conference on Ontologies, Databases and Applications of Semantics for Large Scale Information Systems (ODBASE-2004)*, Larnaca, Cyprus, October 2004, pp. 945–962.

[28] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch, "EDUTELLA: A P2P networking infrastructure based on RDF," in *11th International Conference on World Wide Web (WWW-2002)*, 2002, pp. 604 – 615.

[29] M. A. Rahman, M. Masud, I. Kiringa, and A. El-Saddik, "Bi-Level Mapping: Combining Schema and Data Level Heterogeneity in Peer Data Sharing Systems," in *3rd Alberto Mendelzon International Workshop on Foundations of Data Management*, 2009.

[30] L. Serafini and C. Ghidini, "Using Wrapper Agents to Answer Queries in Distributed Information Systems," in *Procs. of the First International Conference on Advances in Information Systems (ADVIS-2000)*. Springer-Verlag, 2000, pp. 331–340.

[31] L. Serafini, F. Giunchiglia, J. Mylopoulos, and P. A. Bernstein, "Local Relational Model: A Logical Formalization of Database Coordination," in *Procs. of the 4th International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-2003)*, 2003, pp. 286–299.

[32] Y. Tzitzikas and C. Meghini, "Ostensive Automatic Schema Mapping for Taxonomy-based Peer-to-Peer Systems," in *Seventh International Workshop on Cooperative Information Agents (CIA-2003)*, Helsinki, Finland, August 2003, pp. 78–92, (Best Paper Award).

[33] Y. Tzitzikas and C. Meghini, "Query Evaluation in Peer-to-Peer Networks of Taxonomy-based Sources," in *19th Int. Conf. on Cooperative Information Systems (CoopIS-2003)*, Catania, Sicily, Italy, November 2003, pp. 263–281.

[34] Y. Tzitzikas, C. Meghini, and N. Spyratos, "Taxonomy-based Conceptual Modeling for Peer-to-Peer Networks," in *22th Int. Conf. on Conceptual Modeling (ER-2003)*, October 2003, pp. 446–460.

APPENDICES: COMPLETION OF THE EXAMPLE

We resume the example from the processing of the message P_a:TELL$(q3, I(a3))$.

- P_a:TELL$(q3, I(a3))$

  TELL finds the object in the log and updates it. The old log on P_a was:

P_a log
$(P_a, q1, t, 2, \{\{q2, q3\}\})$

  The new log is:

P_a log
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$

- P_a:ASK$(P_a, q2, a2, \{t, a2\})$

  Since there are two incoming hyperedges in $a2$, both in P_b, ASK enqueues 3 ASK messages to P_b, one for each involved term:

  – P_b:ASK$(P_a, q4, b3, \{t, a2, b3\})$
  – P_b:ASK$(P_a, q5, b1, \{t, a2, b1\})$
  – P_b:ASK$(P_a, q6, b2, \{t, a2, b2\})$

  It then persists the corresponding log object. The new log is:

P_a log
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$
$(P_a, q2, a2, 3, \{\{q4\}, \{q5, q6\}\})$

  and issues the 3 enqueued messages.

- P_b:ASK$(P_a, q4, b3, \{t, a2, b3\})$

  Since there are no incoming hyperedges in $b3$, the message P_a:TELL$(q4, I(b3))$ is produced.

- P_a:TELL$(q4, I(b3))$

  TELL finds the object in the log and updates it. The updated log is:

P_a
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$
$(P_a, q2, a2, 2, \{\{I(b3)\}, \{q5, q6\}\})$

- P_b:ASK$(P_a, q5, b1, \{t, a2, b1\})$

  Since there are two incoming hyperedges in $b1$, ASK enqueues 2 ASK messages to P_c, one for each involved term:

  - P_c:ASK$(P_b, q7, c1, \{t, a2, b1, c1\})$
  - P_c:ASK$(P_b, q8, c2, \{t, a2, b1, c2\})$

  It then persists the corresponding log object. The log is now:

P_b log
$(P_a, q5, b1, 2, \{\{q7\}, \{q8\}\})$

- P_b:ASK$(P_a, q6, b2, \{t, a2, b2\})$

  Since there is one incoming hyperedge in $b2$, ASK enqueues 2 ASK messages to P_c, one for each involved term:

  - P_c:ASK$(P_b, q9, c2, \{t, a2, b2, c2\})$
  - P_c:ASK$(P_b, q10, c3, \{t, a2, b2, c3\})$

  It then persists the corresponding log object. The log is now:

P_b log
$(P_a, q5, b1, 2, \{\{q7\}, \{q8\}\})$
$(P_a, q6, b2, 2, \{\{q9, q10\}\})$

- P_c:ASK$(P_b, q7, c1, \{t, a2, b1, c1\})$

  Since there are no incoming hyperedges in $c1$, ASK generates P_b:TELL$(q7, I(c1))$.

- P_b:TELL$(q7, I(c1))$

  TELL finds the object in the log and updates it. The new log is:

P_b log
$(P_a, q5, b1, 1, \{\{I(c1)\}, \{q8\}\})$
$(P_a, q6, b2, 2, \{\{q9, q10\}\})$

- P_c:ASK$(P_b, q8, c2, \{t, a2, b1, c2\})$

  Since there is one incoming hyperedge in $c2$ but its tail has a non-empty intersection with the set of visited terms, just a TELL message results: P_b:TELL$(q8, I(c2))$.

- P_b:TELL$(q8, I(c2))$

  TELL finds the object in the log and updates it. The new log is:

P_b log
$(P_a, q5, b1, 0, \{\{I(c1)\}, \{I(c2)\}\})$
$(P_a, q6, b2, 2, \{\{q9, q10\}\})$

  There are no more open calls in the updated log object, therefore the answer to the query $q5$ can be computed as $I(c1) \cup I(c2)$. Then the object is deleted permanently from the log and the message P_a:TELL$(q5, I(b1) \cup I(c1) \cup I(c2))$ is issued.

- P_a:TELL$(q5, I(b1) \cup I(c1) \cup I(c2))$

  TELL finds the object in the log and updates it. The new log is:

P_a log
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$
$(P_a, q2, a2, 1, \{\{I(b3)\}, \{I(b1) \cup I(c1) \cup I(c2), q6\}\})$

- P_c:ASK$(P_b, q9, c2, \{t, a2, b2, c2\})$

  Since there is one incoming hyperedge in $c2$, ASK enqueues 2 ASK messages to P_b, one for each involved term:

  - P_b:ASK$(P_c, q11, b1, \{t, a2, b2, c2, b1\})$
  - P_b:ASK$(P_c, q12, b3, \{t, a2, b2, c2, b3\})$

  It then persists the corresponding log object. The updated log is:

P_c log
$(P_b, q9, c2, 2, \{\{q11, q12\}\})$

- P_c:ASK$(P_b, q10, c3, \{t, a2, b2, c3\})$

  Since there are no incoming hyperedges in $c3$ a TELL message results: P_b:TELL$(q10, I(c3))$.

- P_b:TELL$(q10, I(c3))$

  TELL finds the object in the log and updates it. The updated log is:

P_b log
$(P_a, q6, b2, 1, \{\{q9, I(c3)\}\})$

- P_b:ASK$(P_c, q11, b1, \{t, a2, b2, c2, b1\})$

  There are two incoming hyperedges in $b1$, but the one having $c2$ in the tail generates no ASK messages. The only ASK enqueued is therefore:

  - P_c:ASK$(P_b, q13, c1, \{t, a2, b2, c2, b1, c1\})$

It then persists the corresponding log object. The updated log is:

P_b log
$(P_a, q6, b2, 1, \{\{q9, I(c3)\}\})$
$(P_c, q11, b1, 1, \{\{q13\}\})$

- P_b:ASK$(P_c, q12, b3, \{t, a2, b2, c2, b3\})$

  Since there are no incoming hyperedges in $b3$, it results: P_c:TELL$(q12, I(b3))$.

- P_c:TELL$(q12, I(b3))$

  TELL finds the object in the log and updates it. The new log is:

P_c log
$(P_b, q9, c2, 1, \{\{q11, I(b3)\}\})$

- P_c:ASK$(P_b, q13, c1, \{t, a2, b2, c2, b1, c1\})$

  Since there are no incoming hyperedges in $c1$, ASK issues P_b:TELL$(q13, I(c1))$.

- P_b:TELL$(q13, I(c1))$

  TELL finds the object in the log and updates it. The new log is:

P_b log
$(P_a, q6, b2, 1, \{\{q9, I(c3)\}\})$
$(P_c, q11, b1, 0, \{\{I(c1)\}\})$

  There are no more open calls in the updated log object, therefore the answer to the query $q11$ can be computed as $I(c1)$. Then the object is permanently deleted from the log and the message P_c:TELL$(q11, I(b1) \cup I(c1))$ is issued.

- P_c:TELL$(q11, I(b1) \cup I(c1))$

  TELL finds the object in the log and updates it. The updated log is:

P_c log
$(P_b, q9, c2, 0, \{\{I(b1) \cup I(c1), I(b3)\}\})$

  There are no more open calls in the updated log object, therefore the answer to the query $q9$ can be computed. Then the object is permanently deleted from the log and the message P_b:TELL$(q9, [(I(b1) \cup I(c1)) \cap I(b3)] \cup I(c2))$ is issued.

- P_b:TELL$(q9, [(I(b1) \cup I(c1)) \cap I(b3)] \cup I(c2))$

  TELL finds the object in the log and updates it. The updated log is:

P_b log
$(P_a, q6, b2, 0, \{\{[(I(b1) \cup I(c1)) \cap I(b3)] \cup I(c2), I(c3)\}\})$

  There are no more open calls in the updated log object, therefore the answer to the query $q6$ can be computed. Then the object is permanently deleted from the log and the message P_a:TELL$(q6, [X \cap I(c3)] \cup I(b2))$ is issued, where

  $$X = [(I(b1) \cup I(c1)) \cap I(b3)] \cup I(c2)$$

- P_a:TELL$(q6, [X \cap I(c3)] \cup I(b2))$

  TELL finds the object in the log and updates it. The updated log is:

P_a log
$(P_a, q1, t, 1, \{\{q2, I(a3)\}\})$
$(P_a, q2, a2, 0, \{\{I(b3)\}, \{I(b1) \cup I(c1) \cup I(c2), [X \cap I(c3)] \cup I(b2)\}\})$

  There are no more open calls in the updated log object, therefore the answer to the query $q2$ can be computed. Then the object is permanently deleted from the log and the message P_a:TELL$(q2, I(a2) \cup I(b3) \cup (Y \cap Z)$ is issued, where

  $$\begin{aligned} Y &= I(b1) \cup I(c1) \cup I(c2) \\ Z &= [X \cap I(c3)] \cup I(b2) \end{aligned}$$

- P_a:TELL$(q2, I(a2) \cup I(b3) \cup (Y \cap Z))$

  TELL finds the object in the log and updates it. The new log is:

P_a log
$(P_a, q1, t, 0, \{\{I(a3) \cap (I(a2) \cup I(b3) \cup (Y \cap Z))\}\})$

  There are no more open calls in the updated object and $q1 \notin T_{P_a}$. Therefore, $q1$ must be a user (external) query.

  The QUERY procedure will realize that $q1$ is complete, and return the answer to the user, thus concluding query evaluation.

AUTHOR BIOGRAPHIES

Dr. Carlo Meghini graduated in Computer Science at the University of Pisa in 1979 and since 1984 he is a researcher at ISTI CNR. He is a member of the Networked Multimedia Information System laboratory, where he leads the Digital Library group (http://nemis.isti.cnr.it/groups/digital-libraries). His areas of interest are digital libraries and semantic technologies. He is involved in European projects since 1986, in the areas of Office Automation, Information Retrieval, Digital Libraries and Digital Preservation. From 2007 he is involved in the making of Europeana, the European digital library (www.europeana.eu), taking care of the scientific aspects of the project. He is currently coordinating the Coordination Action PRELIDA on the Preservation of Linked Data. He has published more than 90 scientific papers in international journals, books and conferences.

Dr. Anastasia Analyti earned a B.S. degree in Mathematics from University of Athens, Greece and a M.S. and Ph.D. degree in Computer Science from Michigan State University, USA. She worked as a visiting professor at the Department of Computer Science, University of Crete, and at the Department of Electronic and Computer Engineering, Technical University of Crete. Since 1995, she is a principal researcher at the Information Systems Lab of the Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH-ICS). Her current interests include the Semantic Web, conceptual modeling, faceted metadata and semantics, rules for the Semantic Web, bio-medical ontologies and systems, contextual organization of information, contextual web-ontology languages, and integration of information. She has participated in several projects and has published over 75 papers in refereed journals and conferences.