A distributed algorithm for robust data sharing and updates in P2P database networks

Enrico Franconi[1], Gabriel Kuper[2], Andrei Lopatenko[1,3], and Ilya Zaihrayeu[2]

[1] Free University of Bozen–Bolzano, Faculty of Computer Science, Italy, franconi@inf.unibz.it, alopatenko@unibz.it
[2] University of Trento, DIT, Italy, kuper@acm.org, ilya@dit.unitn.it
[3] University of Manchester, Department of Computer Science, UK

Abstract. In this paper we thoroughly analyze a distributed procedure for the problem of local database update in a network of database peers, useful for data exchange scenarios. The algorithm supports dynamic networks: even if nodes and coordination rules appear or disappear during the computation, the proposed algorithm will eventually terminate with a sound and complete result.

1 Introduction

In the paper [Franconi *et al.*, 2003] we introduced a general logical and computational characterization of peer-to-peer (P2P) database systems. We first defined a precise model-theoretic semantics of a P2P system, which allows for local inconsistency handling. We then characterized the general computational properties for the problem of answering queries to such a P2P system. Finally, we devised tight complexity bounds and distributed procedures in few relevant special cases. The basic principles of the characterization given in [Franconi *et al.*, 2003] are: (a) the role of the coordination formulas between nodes is for data migration (as opposed to the role of logical constraints in classical data integration systems); (b) computation is delegated to single nodes (local computation); (c) the topology of the network may dynamically change; (d) local inconsistency does not propagate; (e) computational complexity can be low.

In this paper we thoroughly analyze a *distributed procedure* for the problem of local database update in a network of database peers, as defined in [Franconi *et al.*, 2003]. The problem of *local database update* is different from the problem of query answering. Given a P2P database system, the answer to a local query may involve data that is distributed in the network, thus requiring the participation of all nodes at query time to propagate in the direction of the query node the relevant data for the answer, taking into account the (possibly cyclic) coordination rules bridging the nodes. On the other hand, given a P2P database system, a "batch" update algorithm will be such that all the nodes consistently and optimally propagate all the relevant data to their neighbors, allowing for subsequent local queries to be answered locally within a node, without fetching data from

This work has been partially supported by the EU projects Sewasie, KnowledgeWeb, and Interop.

other nodes at query time. The update problem has been considered important by the P2P literature; most notably, recent papers focused on the importance of *data exchange* and *materialization* for a P2P network [Fagin *et al.*, 2003; Daswani *et al.*, 2003].

Relevant work in semantically well founded P2P systems includes [Halevy *et al.*, 2003], which describes an algorithm for acyclic P2P systems using classical (first-order logic) semantics. The acyclic case is relatively simple – a query is propagated through the network until it reaches the leaves of the network. The work in [Calvanese *et al.*, 2003] uses a notion of semantics similar to the semantics introduced in [Franconi *et al.*, 2003], but it describes only a global algorithm, that assumes a central node where all computation is performed. The paper [Serafini and Ghidini, 2000] describes a local algorithm to compute query answers, but it does not allow for existential variables in the head of the coordination rules. The algorithm we present in this paper supports such variables, in a similar fashion to the global algorithm of [Calvanese *et al.*, 2003].

The algorithm presented in this paper supports *dynamic* networks: even if nodes and coordination rules appear or disappear during the computation, the proposed algorithm will eventually terminate with a sound and complete result (under appropriate definitions of the latter). In addition, our algorithm is based on an asynchronous model of communications (while also supporting a synchronous alternative), which means that answering a query, and reaching the fix-point, may be faster at expense of an increase of the number of messages in the network.

2 P2P database systems

Our P2P framework is based on the logical model of [Franconi *et al.*, 2003].

Definition 1 (Local database) *Let I be a nonempty finite set of indexes $\{1, 2, \ldots, n\}$, and C be a set of constants. For each pair of distinct i, $j \in I$, let L_i be a first-order logic without function symbols, with signature disjoint from L_j but for the shared constants C. A* local database *DB_i is a theory on the first order language L_i.*

Nodes are interconnected by means of coordination rules. A coordination rule allows a node i to fetch data from its neighbor nodes $j_1, \ldots, j_m$.

Definition 2 (Coordination rule) *A* coordination rule *is an expression of the form*

$$j_1 : b_1(\mathbf{x}_1, \mathbf{y}_1) \wedge \cdots \wedge j_k : b_k(\mathbf{x}_k, \mathbf{y}_k) \Rightarrow i : h(\mathbf{x})$$

where $j_1, \ldots, j_k, i$ are distinct indices, each $b_l(\mathbf{x}_l, \mathbf{y}_l)$ is a formula of L_{j_l}, and $h(\mathbf{x})$ is a formula of L_i, and $\mathbf{x} = \mathbf{x}_1 \cup \cdots \cup \mathbf{x}_k$.

Note that we are making the simplifying assumption that the equal constants mentioned in the various nodes refer to equal objects, i.e., that they play the role of URIs (Uniform Resource Identifiers). Other approaches consider *domain relations* to map objects between different nodes [Serafini *et al.*, 2003], and we plan to consider such extensions in future work.

A P2P system is just the collection of nodes interconnected by the rules.

Definition 3 (P2P system) *A* peer-to-peer (P2P) system *is a tuple of the form $MDB = \langle LDB, CR \rangle$, where $LDB = \{DB_1, \cdots, DB_n\}$ is the set of local databases, and CR is the set of coordination rules.*

A user accesses the information hold by a P2P system by formulating a query at a specific node.

Definition 4 (Query) *A* local query *is a first order formula in the language of one of the local databases DB_i.*

The semantics of a P2P system and of queries is defined in [Franconi *et al.*, 2003]. In this paper, we assume that all nodes are relational databases; coordination rules may contain conjunctive queries in both the head and body (without any safety assumption and possibly with built-in predicates). Under these assumptions computing of answers is reducible to data fetching [Franconi *et al.*, 2003; Calvanese *et al.*, 2003].

To describe the P2P networks we introduce the notion of a *dependency edge* between nodes of a P2P network.

Definition 5 *There is a* dependency edge *from a node i to node j, if there is a coordination rule with head at node i and body at node j.*

Note that the direction of a dependency edges is the opposite to that of the rules. The direction of a rule is the direction in which data is transfered, whereas the dependency edge has the opposite orientation. In this paper we use $\mathcal{MDB}$ to denote a P2P system, using terms such as *P2P system* or *a network*; please note that we consider the general case when the network is *cyclic*. $\mathcal{I}$ is used to denote a set of all nodes in given $\mathcal{MDB}$, $\mathcal{C}$ denotes the set of all coordination rules, and $\mathcal{L}$ the set dependency edges between nodes in a network derived from $\mathcal{C}$. Subsets of $\mathcal{I}$ are denoted by $\mathcal{A}$. We assume that $\mathcal{I}$, $\mathcal{L}$, and $\mathcal{C}$ are always finite sets.

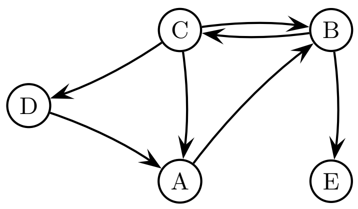
Definition 6 *A dependency path for a node i is a path $\langle i_1, i_2, \ldots, i_n \rangle$ of dependency edges, such that 1) $i_1 = i$; 2) $\langle i_1, \ldots, i_{n-1} \rangle$ is a simple path (no one node met twice).*

Definition 7 *A maximal dependency path for a node i is a dependency path such that if we add any node to the path, the result will not be a dependency path. In this paper, when we describe dependency paths for a node i, we omit the first node (i).*

As an example, consider a P2P system with the follow schemas and rules:

$$\begin{array}{ll}
A : a(X,Y) & r1 : E : e(X,Y) \rightarrow B : b(X,Y) \\
B : b(X,Y) & r2 : B : b(X,Y), b(Y), Z \rightarrow C : c(X,Z) \\
C : c(X,Y), f(X) & r3 : C : c(X,Y), c(Y,Z) \rightarrow B : b(X,Z) \\
D : d(X,Y) & r4 : B : b(X,Y), b(X,Z), X \neq Z \rightarrow A : a(X,Y) \\
E : e(X,Y) & r5 : A : a(X,Y) \rightarrow C : f(X) \\
 & r6 : A : a(X,Y) \rightarrow D : d(Y,X) \\
 & r7 : D : D(X,Y), D(Y,Z) \rightarrow C : c(X,Y)
\end{array}$$

The dependency edges and the maximal dependency paths for the example are:



#	path	#	path	#	path	#	path
A	ABCA	B	BE	C	BE	D	ABE
A	ABE	B	BCAB	C	BC	D	ABCD
A	ABCB	B	BCB	C	DABC	D	ABCB
A	ABDA	B	BCDAB	C	ABC	D	ABCA
				C	ABE		

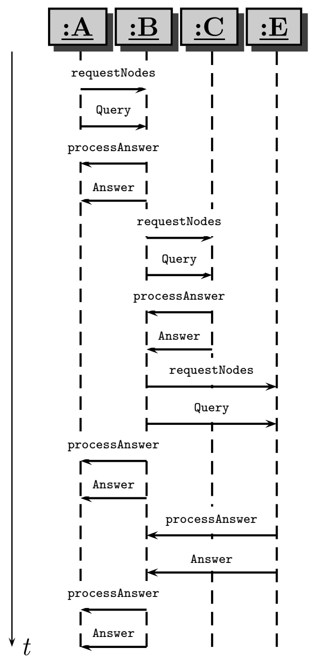
3 The distributed update algorithm

The problem we want to solve is to propagate to all nodes all the information which is distributed among all the databases, so that queries can be answered locally. The main idea behind the distributed update algorithm is that nodes update their local databases by importing data from their neighbors (called acquaintances) using the definitions of coordination rules w.r.t. these neighbors. The global update procedure is started by some super-node, which sends global update requests to its acquaintances, these acquaintances propagate these requests to their acquaintances, and so on. The work of the algorithm seems easy until we have loops in the dependency paths. In this case, query results from some node can go through a chain of nodes and return to the given node. In order to avoid infinite loops in propagation of already computed data in the system, node N stops propagating of some result set R iff a) N is contained in the path R has passed through, and b) there is no new data in R for N. In this case we will say that node N reaches a fix-point. This procedure can be proved to be sound and complete with respect the semantics presented in [Franconi *et al.*, 2003]. The algorithm presented here does not present possible optimization; these exploit the knowledge of specific topological structures, and allow for more fine grained queries to acquaintances and answer from acquaintances (delta optimization) in order to minimize data transfer and duplication.

It is assumed that when a node joins the P2P network it knows only its acquaintances. Therefore, the first step of the algorithm is to let each node know the network topology. Each node first looks for the set of its maximal dependency paths (Topology discovery algorithm). At the end of this first phase, all nodes are aware of the relevant part of the network topology - i.e., each node will know about all the maximal dependency paths starting from it; in the P2P literature [Hellerstein, 2003] it is assumed reasonable if a node is aware of *log(n)* 'neighbor' nodes, where n is a size of network. The topology discovery is initiated by a "super-peer", which should be selected to optimize the distribution of messages during discovery time; a super-peer do not have any other property differentiating it from other nodes. It is possible that the topology changes while the discovery process is ongoing; in Section 4 we analyze this case.

The algorithm for the actual database update is the same as the one for the topology discovery, with two main differences: (1) in the discovery phase the topology discovery algorithm stops when a node is reached twice, while the update algorithm has to continue the computation until a fix-point is reached; and

(2) the discovery algorithm does not perform actual database queries and propagation. Both phases of the algorithm are executed asynchronously in parallel; we use the notation $\text{NAME\_OF\_FUNCTION}_i$ to mean the execution of that function at node i. In the following you can see a sample execution of the algorithm for our example P2P system:



Each node is given the following data structure:

- ***state***$_d$, a variable that describes the state of knowledge about the network. It can have the value **discovery**, that means that the discovery process is undergoing and the knowledge is incomplete, or **closed**, when the knowledge about the network is complete. Initially it is undefined.
- ***state***$_u$, a variable that describes the status of the data at a node. It has the value **open** when the node has no complete data, or **closed** when the node has reached the fix-point.
- ***finished***, a boolean variable indicating that the network discovery through the node is finished.
- ***Rules(rule, node, flag)***, a relation that describes the set of coordination rules which have the node as target. The attributes of this relation are *rule*, the id of a rule, *node*, the id of the source node for that rule, and a *flag* (see the algorithm). We assume that initially each node knows all rules of which it is a target.
- ***Paths(path, flag, closed)***, a relation that describes the maximal dependency paths for the node. Its attributes are: *path*, a string that represents the path, and *flag*, *closed* (see the algorithm). We assume that initially the relation is empty.

– ***Edges(source, target)***, a relation that describes all dependency relations in the part of the network reachable from the node.
– ***ID*** is the identifier of the node, assumed to be unique in the network.
– ***owner***, an array which contains pairs of node IDs, the first being a node ID on behalf of which the node is searching for data, and the second being the node ID which sent the request.

In the algorithm we use the function **id(rule)** to take the ID of a rule's ID and return the ID of the source node of the rule.

A1: Network topology discovery.
The algorithm is executed by the 'super-peer' which starts the process of network discovery. All the other nodes will only run the QUERY and PROCESSANSWER algorithms when they are requested.

```
DISCOVER($Rules$, $ID$)
if $|Rules| == 0$
        $state_d = closed$ ; $Paths = \emptyset$ ; return
    if $state_d == \perp$
      $state_d = discovery$
      foreach $r \in Rules$
              REQUESTNODES$_{id(r)}$($ID$, $ID$)
          $owner = owner \cup \langle \emptyset, ID \rangle$
```

A2: Topology discovery: process request
Process the request sent by another node; ID_s is an ID of the node sending request, ID_o is the ID of the node on behalf of which the request is sent.

```
REQUESTNODES($ID_s, ID_o$)
if $Rules == \emptyset$
        $state_d = closed$ ; $finished = true$
    if $ID_o \notin \pi_2(owner)$
      foreach $r \in Rules$
          REQUESTNODES$_{id(r)}$($ID, ID_o$)
      else
              $finished = true$
          $owner = owner \cup \langle ID_s, ID_o \rangle$
          PROCESSANSWER$_{ID_s}$($ID_o$, $Edges \cup \langle ID, ID_s \rangle$, $state_d$, finished)
```

A3: Topology discovery: process the answer from another node
ID_o is a ID of a node on behalf of which discovery is done, *set* is a set of answers (dependency edges between nodes), and *state* is the answer completeness status, *status* is an indicator that discovery in a given branch is finished, **me** is an ID of the rule to which the current answer relates.

```
PROCESSANSWER($ID_o, set, state, status$)
$Edges = Edges \cup set$
if $state ==$ closed
        $update\ Rules\ set\ flag = true\ where\ rule = me$
    if $status ==$ true
          $update\ Rules\ set\ closed = true\ where\ rule = me$
    if $\forall Rulesflag == true$
        $state_d = closed$
      if $\forall Rulesfinished == true$
          $finished = true$
        if $ID == ID_o$

            if $\forall Rulesfinished == true$
              $stated_d = closed$
             foreach (
              $\langle ID_s, ID_o \rangle \in owner$)
                  processAnswer$_{ID_s}$($ID_o$, $Edges$, $state_d$, finished)
```

A4: Database update: process query sent by another node
ID_s is the ID of the node which did send the request, Q is a query (the head of the coordination formula), SN is a sequence of nodes' IDs describing a path for the query evaluation.

```
Query($ID_s, Q, SN$)
QA = Compute-local-answer(Q)
$Answer_{ID_s}$(ID, QA, ID + SN, $state_u$)
$owner = owner \cup ID_s, first(SN)$
if $state_u == open \wedge ID \notin SN$
    foreach $r \in Rules$
        $Query_{id(r)}$(ID, Query(r), ID + SN)
```

A5: Database update: Process the answer sent by another node
ID is an ID of the node which sent the answer, QA is the answer as a set of tuples, SN is a sequence of nodes' IDs, state is a flag indicating if the answer is complete.

```
Answer($ID, QA, SN, state$)
update = UpdateLocalData(rule, QA)
if state == complete
    update Rules set flag = true where rule = rule(Q)
  if $update = \emptyset$
      update Paths set flag = true where path = SN
    if $update \neq \emptyset$
        update Paths set flag = false where path = SN
      if $\forall t \in Rules\pi_{flag}(t) == true$
          $state_u = closed$
        if $\forall t \in Paths\pi_{flag}(t) == true$
            $state_u = closed$
          foreach $node \in \pi_1(owner_u)$
              QA = ComputeAnswer()
              $Answer_{node}$(ID, QA, SN, $state_u$)
```

A6: Database update: local update algorithm.
Updates the database to make it consistent with the view extensions (i.e., the rules) and the original content. Rule is the identifier of a rule, QA is a set of tuples.

```
UpdateLocalData($rule, QA$)
if (
  $QA \not\equiv \emptyset$)
    foreach (
      $tuple\ t \in QA$) foreach (
        $relation\ R \in definition(View_{rule})$)
        if $\pi_R(t) \neg \in R$
          insert $(\pi_R(t))$ into R with new values for existential
```

The following theorem states the correctness of the proposed update algorithm, and its complexity.

Theorem 1 *1. (Soundness and completeness) A node reaches the state closed iff the algorithm has reached the fix-point at this node;*
2. (Termination) Every node of the network eventually reaches the state closed;
3. (Complexity) The complexity of the algorithm database update at each node is 2EXPTIME in the number of nodes.

4 Dynamic behavior of the P2P network

One of the distinctive characteristics of P2P systems is that the network can vary dynamically. Assume that the network $\mathcal{MDB}$ consist initially of a set of nodes $\mathcal{I}$, and that $\mathcal{C}$ is an initial set of coordination rules with $\mathcal{L}$ being the initial set of dependency edges. We model network dynamicity by adding/removing coordination rules between nodes, and therefore deletion of a node is modeled by deleting all coordination rules that relate to this node. With respect to query answering adding/removing nodes with coordination rules is easily seen to be equivalent to the assumption that all nodes are present from the start, and that only coordination rules are changed.

We define an atomic network change operation as follows.

- *addLink(i,j,rule,id)*: add the coordination rule *rule* from node *j* (the body) to node *i* (the head). *id* is the name of a rule, which should be unique for a given pair of nodes.
- *deleteLink(i,j,id)*: delete the coordination rule *id* between nodes *i* and *j*

Definition 8 *1. A* change $\mathbf{U}$ *of a network* $\mathcal{MDB}$ *is a sequence of atomic change operations over* $\mathcal{MDB}$.
2. A finite change *of a network is a finite sequence of atomic changes.*
3. An initial subchange $\mathbf{U}_1$ *of a change* $\mathbf{U}$ *is a initial prefix of* $\mathbf{U}$
4. A subchange $\mathbf{U}_\mathcal{A}$ *of* $\mathbf{U}$ *in respect to* $\mathcal{A} \subset \mathcal{I}$ *is a set of atomic operations of* $\mathbf{U}$, *relevant to* $\mathcal{A}$ *and ordered with the same order as in* $\mathbf{U}$

We assume that in the case of atomic change the network will be notified about the change in the following cases:

1. in case of *addLink(i,j,rule,id)*, the node *i* (which will be able to fetch data by this rule) gets a notification; *addRule(i, j, rule, id)*

2. in case of *deleteLink(i,j,id)*, the node i (which will be unable to feth data by this rule) gets a notification. *deleteRule(i, j, id)*

Definition 9 *1. A* sound answer *of a query Q in a network subject to runtime changes, is an answer to the query that is included in the result that we would obtain if we executed all the addLink statements before running Q, and did not execute the deleteLink statements at all.*

2. A complete answer *of a query Q in a network subject to runtime changes, is an answer to the query that contains the result that we would obtain if we executed all the deleteLink statements before running Q, and did not execute the addLink statements at all.*

The basic idea behind this definition is that we cannot know in advance what the state of the database will be at termination time. Therefore, in the definition we require that a sound and/or complete answer will be classically sound and/or complete with respect to the part of the database that is *unchanged*. The result with respect to the part that is changed will depend on the order and timing of the execution of the changes. In this sense, the answer to a query in a network subject to "small" changes will be still meaningful with respect to the majority of the data that resides in the stable parts of the network. The following theorem states that our update algorithm behaves well with respect to change.

Theorem 2 *1. For a finite change of a network, the update algorithm will terminate, and it gives sound and complete answers to queries in the network subject to runtime changes.*

2. In the case of an infinite change to the network, the update algorithm may not terminate.

However, in general we cannot assume that a network change is finite. In the general case, therefore, the nodes in the network may never reach the fix-point – or at least, we may not be able to show that they have reached a fix-point. We now give a condition on when a subset of nodes can reach a fix-point, even under infinite change of the whole network.

Definition 10 *1. A set of nodes $\mathcal{A}_1$ is separated from a set of nodes $\mathcal{A}_2$ in a P2P network $\mathfrak{I}$ if there is no dependency path from any node in $\mathcal{A}_1$ that involves a node in $\mathcal{A}_2$.*

2. A set of nodes $\mathcal{A}_1$ is separated from a set of nodes $\mathcal{A}_2$ in P2P network $\mathfrak{I}$ with respect to a change $\mathbf{U}$ if for any subchange of $\mathbf{U}$ there is no dependency path from a node in $\mathcal{A}_1$ involving a node in $\mathcal{A}_2$ in the network we obtain by applying that subchange.

Theorem 3 *If, for a network $\mathfrak{I}$, a set of nodes $\mathcal{A}$ is separated from $(\mathfrak{I} \setminus \mathcal{A})$ with respect to a (possibly infinite) change $\mathbf{U}$ over $\mathfrak{I}$, and $\mathbf{U}_{\mathcal{A}}$ is finite, then the algorithm, when applied to a node in $\mathcal{A}$, terminates, yielding a sound and complete answer.*

Lemma 4 *For a finite runtime change of the network, the complexity of the update algorithm at each node is in 2EXPTIME with respect to the size of the change.*

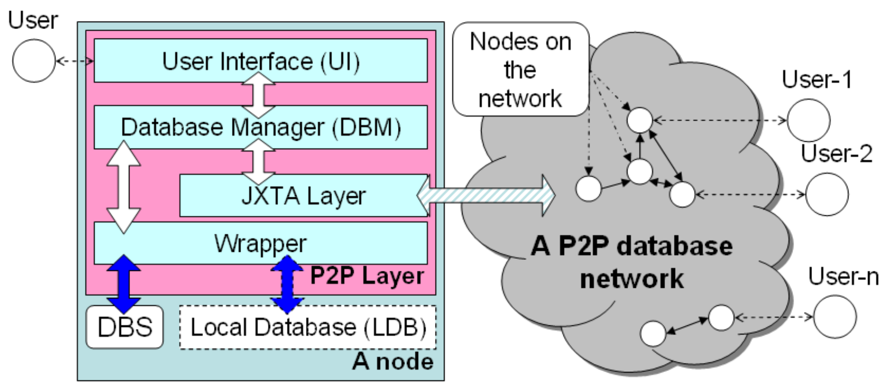


Fig. 1. First level architecture

5 Implementation

We implement database peers on top of *JXTA* [JXTA project,]. JXTA specifies a set of protocols which provide implementation of basic, as well as rather sophisticated P2P functionalities. As basic functionalities we can distinguish: definition of a peer on a network; creation of communication links between peers (called pipes); creation of messages, which can envelope arbitrary data (e.g. code, images, queries); sending messages onto pipes, etc. Examples of more sophisticated functionalities provided by JXTA are: creation of peer groups; specification of services and their implementation on peers; advertising of network resources (i.e. peers, pipes, peer groups, services, etc.) and their discovery in a distributed, decentralized environment. JXTA has a number of advantages for developing P2P applications. It provides IP independent naming space to address peers and other resources, it is independent of system platforms (e.g. Microsoft Windows or UNIX) and networking platforms (e.g. Bluetooth, TCP/IP), it can be run on various devices such as PCs or PDAs, and provides support for handling firewalls and NATs. We have chosen JXTA since it already gives practically all basic building blocks for developing P2P applications and thus allow the developer to concentrate on implementation of specific functionalities a given application is required to provide.

First level logical architecture of a node, inspired by [Bernstein *et al.*, 2002] and [Giunchiglia and Zaihrayeu, 2003], is presented on Figure 1. A node consists of *P2P Layer*, *Local Database* (LDB) and *Database Schema* (DBS). DBS describes part of LDB, which is shared for other nodes. P2P Layer consists of *User Interface* (UI), *Database Manager* (DBM), *JXTA Layer* and *Wrapper*. Nodes connect to a P2P database network by means of connecting to other peer(s), as it is schematically shown on Figure 1 (see the arrow from JXTA Layer to the network and arrows between nodes in the network).

By means of UI users can commence network queries and updates, browse streaming results, start topology discovery procedures, and so on. Among other things, UI allows to control other modules of P2P Layer. For instance, user can modify the set of coordination rules w.r.t. other nodes, define connection details for Wrapper, etc. DBM processes both user queries and queries coming from the network, as well as global and query-dependent update requests. It is also responsible for processing of query results coming both from LDB and the network, as well as for processing of updates results coming from the network. Finally, DBM manages propagation of queries, update requests, query results and update results on the network. JXTA Layer is responsible for all node's activities on the network, such as discovering of previously unknown nodes, creating pipes with other nodes, sending messages containing queries, update requests, query results, etc. Wrapper manages connections to LDB and executes input database manipulation operations. This is a module which is adjusted depending on the underlying database. For instance, when LDB does not support nested queries, then this is the responsibility of Wrapper to provide this support. Yet another task of Wrapper is retrieval and maintenance of DBS.

The LDB rectangle stands for RDBMS. It has dashed border to mean that local database may be absent. Nevertheless DBS must always be specified in order to allow a node to participate on the network. In this situation a given node acts as a mediator for propagating of requests and data, and all required database operations (as join and project) are executed in Wrapper. The DBS rectangle has rounded corners because it represents a repository, where DBS is stored. Arrows between UI and DBM as well as arrows between JXTA Layer, Wrapper and DBM have the same graphical notation because they represent procedure calls between different execution modules. The arrow between JXTA Layer and the network has another notation because it represents communication supported by JXTA. The arrows connecting Wrapper, DBS and LDB have yet another notation because the communication they denote is LDB dependent.

For the purposes of collecting experimental data, each node has an additional statistical module (not shown on Figure 1). This module accumulates information about number of executed queries and updates, total time which was required to answer a certain query or fulfill an update request, volumes of data transferred onto pipes, number of queries received and sent for the same original query (due to different paths and loops), and so on. For the statistical purposes, there is a possibility to make some peer the super-peer on the network. This peer can read coordination rules for *all* peers from a file and broadcast this file to all peers on the network. Once received this file, each peer looks for relevant to it coordination rules, reads them, creates and drops pipes with other nodes, where necessary. Thus, one peer can change the network topology at *run-time*. This is extremely convenient for running multiple experiments on different topologies. Finally, the super-peer can "command" other peers to send to it statistical information accumulated by some moment of time, or reset statistics at all peers.

Current version of the prototype supports both global and query-dependent updates handling, distributed query answering, and implements the topology discovery algorithm. When a node starts, it creates pipes with those nodes, w.r.t. which it has coordination rules, or which have coordination rules w.r.t. the given node. Several coordination rules w.r.t. a given node can use one pipe to send requests and data. If some coordination rules are dropped and a pipe becomes unassigned a coordination rule, then this pipe is also closed.

Preliminary experiments were done to measure the scalability of our approach with respect to a size of the P2P network. We are currently organizing more precise experiments.

Up to 31 nodes participated to the preliminary experiments. The local relational databases are based on DBLP data (`http://dblp.uni-trier.de/xml`) and contained about 20000 records about publications (about 1000 per node), organised in 3 different relational schemas. We considered two different data distributions. In the first one there is no intersection between initial data in neighbor nodes. In the second, there is 50% probability of intersection between initial data in nodes linked by coordination rules; the intersection between data in other nodes is empty. Three types of topologies have been considered: trees, *layered* acyclic graphs, and cliques.

By looking at the execution time and the number of messages exchanged between nodes, the preliminary experiments confirmed the expectation that in the simple topological structures (like the tree and the layered acyclic graphs) the execution time is linear with respect to the depth of the structure.

References

[Bernstein *et al.*, 2002] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. *WebDB*, 2002.

[Calvanese *et al.*, 2003] Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Semantic data integration in p2p systems. In *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2003)*, 2003.

[Daswani *et al.*, 2003] Neil Daswani, Hector Garcia-Molina, and Beverly Yang. Open problems in data-sharing peer-to-peer systems. In *ICDT 2003*, 2003.

[Fagin *et al.*, 2003] Ronald Fagin, Phokion G. Kolaitis, Ren&#233;e J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Proceedings of the 9th International Conference on Database Theory*, pages 207–224. Springer-Verlag, 2003.

[Franconi *et al.*, 2003] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Luciano Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *International Workshop On Databases, Information Systems and Peer-to-Peer Computing*, 2003.

[Giunchiglia and Zaihrayeu, 2003] F. Giunchiglia and I. Zaihrayeu. Implementing database coordination in p2p networks. *DIT technical report # DIT-03-035, the University of Trento, Italy*, November 2003.

[Halevy *et al.*, 2003] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *ICDE*, 2003.

[Hellerstein, 2003] Joseph M. Hellerstein. Toward network data independence. *SIGMOD Rec.*, 32(3):34–40, 2003.

[JXTA project,] JXTA project. see `http://www.jxta.org`.

[Serafini and Ghidini, 2000] Luciano Serafini and Chiara Ghidini. Using wrapper agents to answer queries in distributed information systems. In *Proceedings of the First Biennial Int. Conf. on Advances in Information Systems (ADVIS-2000)*, 2000.

[Serafini *et al.*, 2003] Luciano Serafini, Fausto Giunchiglia, John Mylopoulos, and Philip A. Bernstein. Local relational model: A logical formalization of database coordination. In *CONTEXT 2003*, pages 286–299, 2003.