UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

IMPLEMENTING DATABASE COORDINATION
IN P2P NETWORKS

Fausto Giunchiglia and Ilya Zaihrayeu

November 2003

Technical Report # DIT-03-035

Implementing Database Coordination in P2P Networks

Fausto Giunchiglia and Ilya Zaihrayeu
Dept. of Information and Communication Technology
University of Trento
38050, Povo, Trento, Italy
{fausto, ilya}@dit.unitn.it

Abstract. We are interested in the interaction of databases in Peer-to-Peer (P2P) networks. In this paper we propose a new solution for P2P databases, that we call *database coordination*. We see coordination as managing semantic interdependencies among databases at runtime. We propose a data coordination model where the notions of *Interest Groups* and *Acquaintances* play the most crucial role. Interest groups support the formation of peers according to data models they have in common; and acquaintances allow for peers inter-operation. Finally, we present an architecture supporting database coordination and show how it is implemented on top of JXTA.

1 Introduction

Peer-to-Peer is a networking model where all parties, called *peers* (or nodes), have equivalent capabilities in providing other parties with data and/or services. To cope with these tasks, peers cooperate in a decentralized, distributed manner. P2P paradigms have been successfully implemented in a number of different domains. For instance, ICQ[1] allows its users to exchange messages in a synchronous way; Kazaa[2], the largest P2P file sharing application, allows for uploads/downloads of music, video, program and other files among peers; DataGrid[3] is a European project aimed on developing a distributed computing infrastructure, where peers can contribute their processing power and storage capacities to provide huge processing and storage resources to individuals. Groove[4] allows for the exploitation of human presence at the edges of Internet, namely the creation of a secure workspace where people can communicate with each other, share, discuss and jointly edit files, conduct conferences, and so on.

Despite the fact that P2P networks have a wide range of applications, there is still a niche for potentially successful application domains which have not been yet (largely) explored. One of them, involving both P2P and database technologies, is called *P2P databases*. The key idea is to support databases in

---

[1] http://web.icq.com
[2] http://www.kazaa.com
[3] http://www.eu-datagrid.org
[4] http://www.groove.net

their inter-operation with a modality coherent with the P2P nature. To fulfill this task, each DB peer should be able to share (a part of) its database, send queries to the network, answer and effectively route queries, coming from the network.

Not much work has been done so far on P2P databases. The University of Washington is working on a project called *Piazza* [1]. Piazza allows one to define semantic mappings between pairs of database peers (or among small subsets of peers) by means of mediated schemas, and proposes techniques for using these mappings for query answering. The University of Toronto is developing a project called Hyperion, which aims at the definition of a peer-to-peer data management architecture and the study of viable data integration, exchange and mapping mechanisms for the P2P environment [2]. A project called PeerDB is being developed at the University of Singapore. PeerDB is a P2P system for distributed data sharing. It is focused mostly on the development of the architecture that would support P2P database management, as well as query processing, mainly assisted by agents [3]. Some work on this topic, based on the definition of a new theoretical framework called "Local Relational Model", has also been done at the University of Trento [4].

We propose a new solution to P2P databases, that we call *database coordination*. We see coordination as the ability of peers to effectively manage, at runtime, semantic interdependencies among databases in a decentralized, distributed and collaborative manner. In a dynamic, heterogeneous P2P environment, solutions, relying on centralized query answering paradigms (as in data integration [5]), no longer apply. Some relevant ideas about database coordination are discussed in [6]. Our goal in this paper is to show how these ideas can be implemented inside a concrete architecture, built on top of JXTA [7]. In this paper we concentrate only on query answering. In order to know how updates can be supported in a P2P database network see [8].

This paper is organized as follows. Section 2 introduces the four basic architectural notions of our model. Section 3 explains how these notions can be implemented in JXTA. Section 4 discusses the logical architecture that we propose. Finally, Section 5 gives the conclusions.

2   A model for data coordination

We consider the notion of a *peer* as primitive, and take it to be any device on a network. Each participating peer supplies a source database with its schema, or provide just the schema. In the latter case a peer acts as a mediator in the transitive propagation of queries and query results. Peers are largely autonomous, in particular, in which data they store, in how they describe the data in a schema, and in which other peers they communicate with. We define data coordination in terms of four basic notions. They are: *Interest Groups, Acquaintances, Correspondence Rules* and *Coordination Rules*.

2.1 Interest Groups

In most cases, peers know very little about the *topics* other peers are able to answer queries about. Intuitively, "Hotels in Italy", "Art collections" are all possible topics. A topic can be formalized as a set of keywords, a generic query schema, or as an ontology. An *Interest Group* (or a *Peer Group*) is a group of peers able to answer queries about a certain topic. Instead of sending a query to single peers, a node sends it to one or more interest groups whose topics are relevant to the query. We suppose that queries are also associated with a topic, which can be defined by the user or by the system itself. Thus, a group topic must be general enough to capture a wide range of related queries on one hand, and specific enough to allow for more relevant query answers on the other.

Interest groups are used to compute, for any given input query, a *Query Scope* (QS) – a set of nodes in a peer group whose schemas can be used to answer a particular query. This allows us to reduce the number of messages in the system and to collect *meaningful* answers. Each group has a *Group Manager* (GM), a node that computes query scopes. For query scope computation we partially adopt techniques used in data integration systems, namely Local-As-View (LAV) and Global-As-View (GAV) [5]. Central role is given to the notion of a *Global Schema* (GS) which is an integrated view of the set of local schemas. Queries are posed against GS and, according to the mappings between the local schemas and GS, are *reformulated* with respect to the local schemas. There are two basic ways to define these mappings. If the local schemas are defined in terms of GS then this is the LAV approach. If GS is defined in terms of the local sources then it is GAV. In GAV, query reformulation tasks are rather simple, but if a source changes, or a new one is added, then GS needs to be reconsidered. In LAV, GS does not need to be changed when there is a change in local sources; instead, only the mappings from GS to a particular local schema are affected. A major drawback of LAV is that query reformulation is rather complex [9].

A group manager maintains both GS and the set of mappings from GS to the local schemas of nodes in the group with the only purpose to compute nodes to be included into QS. Mappings are expressed as *conjunctive queries* without comparison predicates [5]. We omit comparison predicates because participating databases may have different domains to represent similar concepts (e.g. prices are in euros in one database and in dollars in another), and therefore a mapping function is required to compare values from different domains. To cope with this problem we use coordination rules and correspondence rules as described below in this section.

The Query Scope computation works as follows. A user submits a query Q_i, which is formulated w.r.t. local schema ls_i of some node. This node sends it to GM with a request to compute QS. In its turn, GM reformulates Q_i to Q_{GS} which is now formulated w.r.t. GS. In order to do this, GM uses the mappings for schema ls_i. Note, that now we treat the mappings as being GAV (i.e. as if ls_i were a global schema and GS were a local one) and therefore query reformulation is easy. Given Q_{GS}, GS and the set of mappings, GM computes QS for Q_i. To fulfil this task, we partially reuse the *bucket algorithm* [9], which was developed

as part of the Information Manifold System [10]. In particular, we do not proceed to the most expensive part of the algorithm, where query containment check and reformulation is done (and therefore we avoid the query reformulation complexity in LAV); and restrict to the part where relevant mappings (and therefore nodes) are defined.

2.2 Acquaintances

Knowing QS is still not enough for a resultful query answering. *Acquaintances* are nodes a node knows about. Moreover, there is a further request that a node must know how to translate an input query into a *specific query*, formulated with respect to the database of an acquainted node. We will call this specific query an *acquaintance query*. A node is able to propagate those queries to an acquaintance, which are equal or contained in the acquaintance query of that acquaintance. A node must also know how to translate backward query results coming from an acquaintance with respect to its local schema. The notion of the acquaintance is not symmetric. The fact that one node is an acquaintance of another does not necessarily mean that the vice versa holds. We define the syntax of acquaintance queries in the form of conjunctive queries. Conjunctive queries can express select-project-join queries. Consider the following example.

Example 1. Think about three nodes A, B and C. Let us suppose that A has a database that stores data about pieces of art; B stores information on paintings, and C keeps data on sculpture works. All three databases have only one relation, shown below:

$A : Art(\underline{AiD}, Name, Year, Author, Desc)$;
$B : Paint(\underline{PiD}, Title, Century, Painter, Notes)$;
$C : Sculpt(\underline{SiD}, Name, Year, Description)$;

It is easy to see similarities between these schemas. Now, imagine that node B is an acquaintance of node A with respect to an acquaintance query $Q^{A \to B}$:

$$Q^{A \to B}(N, Y, A, D) : -Art(AiD, N, Y, A, D),$$

were N, Y, A, D stand for attributes $Name, Year$, etc. Furthermore, A is acquainted also with C with respect to query $Q^{A \to C}$:

$$Q^{A \to C}(N, Y, D) : -Art(AiD, N, Y, A, D)$$

Note, that in $Q^{A \to C}$ the *Author* attribute is absent. This is motivated by the fact that in $C : Sculpt$ an analogous concept is absent, and therefore A can *not* ask queries to C about the author of a sculpture.

An acquaintance query is conceptually different from the topic associated with an interest group. Interest group topics are supposed to be rather general

and capture an approximated domain of the data stored at *all* peers in the group. An acquaintance query is very specific and dependent on the semantic interdependencies between the local schemas of two nodes.

2.3 Correspondence Rules

In most cases, participating databases are semantically heterogeneous, namely, they represent the same concepts differently [11, 12]. To address this problem we introduce the notion of *correspondence rules*. A correspondence rule is a pair where the first element is from one database, and the second is from another. We need to create correspondences between relations, attributes and values. Each acquaintance is associated with one or more correspondence rules. They explain how to translate queries to be sent to and query results received from a particular acquaintance. Consider the example below:

Example 2. Recall the relations of A, B, C. Correspondence rules associated with acquaintance B of A might include the following correspondences:

$Corr_1^{A \to B}$ `A:Art` $\to$ `B:Paint`;
$Corr_2^{A \to B}$ `Art:Year` $\to$ `Paint:Century`;
$Corr_3^{A \to B}$ `Value(Paint:Century) = Int(Value(Art:Year)/100) + 1`;
$Corr_1^{B \to A}$ `Paint:Century` $\to$ `Art:Year`;
$Corr_2^{B \to A}$ `Value(Art:Year) = Value(Paint:Century)*100 - 50`;

Rules of the form $Corr_i^{A \to B}$ are used for query propagation, namely for translation of concepts in `A` into the corresponding concepts in `B`. Rules of the form $Corr_j^{B \to A}$ are used for backward translation of query results coming from `B` to `A`. In particular, $Corr_1^{A \to B}$ defines the correspondence between relation names; $Corr_2^{A \to B}$ shows how attribute names are related. $Corr_3^{A \to B}$ specifies how the value of the year attribute is translated into the value of the corresponding century. While $Corr_1^{B \to A}$ is a backward attribute names correspondence; and $Corr_2^{B \to A}$ is an approximated translation of century into the year value (e.g. the 20th century gives year 1950).

2.4 Coordination Rules

Each acquaintance is associated with a set of *coordination rules*. Coordination rules specify under which condition to propagate a query to a specific acquaintance. It is a complementary mechanism to query scopes, intended to prune irrelevant nodes which might be contained in QS. Irrelevant nodes might be added to a query scope due to the fact that we do not proceed to the second phase of the bucket algorithm or just because the domains of some databases in QS are out of the scope of the query.

We implement coordination rules similarly to how *Event-Condition-Action* (ECA) rules are implemented in active database systems [13]. Some papers

discussing work on coordination rules are [6, 14, 15]. An *Event* can be a basic database manipulation operation as *select*, *insert*, *update* or *delete*. Since we are focused on query answering, we consider only `select` events. One event can trigger several coordination rules of several acquaintances. The *Condition* part can reference a certain property of a query, such as which items are referenced and what values they are given in the `WHERE` clause. By default, the condition contains an item that verifies whether a given query is contained in or equal to the acquaintance query of a particular acquaintance. The *Action* part usually consists of activating the correspondence rules of a given acquaintance for translation, followed by propagation of a particular query to this acquaintance. Correspondence rules may also be called from the `Condition` part to translate data, where necessary. Consider the example below:

Example 3. Recall the databases A, B and C in Example 1. Let us enrich the scenario by restricting the contents of databases B and C. Now, the particularity of B is that it stores only data about ancient paintings, which come from the middle of the 16th century or earlier. In its turn, C stores only recent sculpture arts, which were produced after 1900 AC. A keeps data about both ancient and modern art. Below are simple examples of coordination rules set up at A for acquaintances B and C:

$CR_1^{A \to B}(Q)$
Event:　　`SELECT from Art`
Condition: $(Q \subseteq Q^{A \to B}) \wedge$ ("painting" $\subseteq$ Desc) $\wedge$ (Year $\leq Corr_2^{B \to A}$ (16))
Action:　　Apply $Corr^{A \to B}$ and propagate to `B`
$CR_2^{A \to C}$ (Q)

Event:　　`SELECT from Art`
Condition: $(Q \subseteq Q^{A \to C}) \wedge$ ("sculpture" $\subseteq$ Desc) $\wedge$ (Year $\geq$ 1900)
Action:　　Apply $Corr^{A \to C}$ and propagate to `C`

Coordination rule $CR_1^{A \to B}$ (Q) is triggered when there is a `select` event in `A:Art`. First, the `Condition` part checks whether this query is contained in the acquaintance query for `B`. If the word `"painting"` is found in the string assigned for the `Description` attribute, and the year is less or equal than the one, corresponding to the 16th century (here we use correspondence rule $Corr_2^{B \to A}$ for translation) – then the condition is `true`. And, if the condition is `true`, then the query is translated, applying the correspondence rules of `B` for forward translation, and propagated to `B`. $CR_2^{A \to C}$ (Q) works analogously. A query to `A:Art` triggers *both* coordination rules. But, depending on the year and the kind of the piece of art being searched for, the query is propagated either to `B` or to `C`, or is not propagated at all.

Coordination rules are the main mechanism for the *transitive propagation* of queries through a chain of nodes. At each node, an incoming query may trigger one or more coordination rules that can lead to consequent propagation.

In practice, coordination rules can be implemented as triggers, if the database management system supports them. Otherwise they can be implemented in the P2P software, as it is described in section 4.

2.5 Database coordination

Let us see how the four basic notions discussed above allow us to implement database coordination. Interest groups allow us to gather peers according to some particular topic, thus increasing the relevance of query answers. Acquaintances are links between peers, parameterized by the acquaintance queries. For any given user query and a node, they provide a set of propagation paths from that node to other nodes in the interest group. Correspondence rules ensure proper information flow along these paths, avoiding distortion and/or loss of data. Coordination rules define query propagation policies along these paths. Finally, group managers allow us to restrict query propagation to the most relevant nodes (with respect to a query) by defining query scopes.

All these notions are used in a query propagation algorithm. Its main idea is that, given a query and a query scope, a node tries to propagate the query to all its acquaintances from the query scope in accordance with the coordination and correspondence rules. The group manager of a given interest group determines when a query propagation is actually complete. To do this, each node reports to GM which acquaintances it propagated a query to. GM keeps this information and starts counting these nodes as *boundary* in the query propagation. If a boundary node propagates further, it is no longer considered as a boundary. If no propagation took place from a boundary node, then it sends to GM a ``no propagation acknowledgement''. A query propagation is said to be complete, when ``no propagation acknowledgements'' are received from all boundary nodes. When this happens, GM reports this fact to the node which asked to compute a query scope.

3   Implementing data coordination in JXTA

We concentrate on the implementation of peers, interest groups and acquaintances. We do not consider coordination rules and correspondence rules, which must be implemented in the application software.

Let us briefly describe the key JXTA notions. *JXTA peers* are devices which implement one or more JXTA protocols. JXTA peers allow their users to provide services for other peers and consume services provided by other peers. *Peer Groups* is the central concept to all aspects of the JXTA platform. A JXTA peer group is a *collection of peers* which agree on providing a *common set of services*. Peer groups form a hierarchical parent-child relationship, where each group has a single parent. JXTA protocols describe how peers may publish, discover, join, and monitor peer groups, but they do not dictate when or why peer groups are created. *Services* are the main underlying mechanisms used to implement

peer groups. They define the functionalities, the peers of a given group possess. Services fall into two categories: *peer services* and *peer group services*. The core services include: *Discovery Service*, *Membership Service* and *Pipe Service*. The Discovery Service allows peers to locate and publish information about the network resources. The Membership Service is used by current members of a peer group to reject or accept a new group membership application. The Pipe Service allows peers to create communication links (pipes) with nodes from the same group. New peer groups may include (a subset of) the core services as well as *custom services*. Custom services allow for the creation of peer groups which will provide their peers with desirable functionality. *Pipes* are the main transport mechanisms, available through the Pipe Service. Pipes are used by peers to send messages from one to another. The pipe *endpoints* are referred as the *input pipe* (the receiving end) and as the *output pipe* (the sending end). Pipe endpoints correspond to available peer network interfaces (e.g., TCP port and IP address). Using the same pipe advertisement *two* nodes can create input pipe at one end and output pipe at another, thus making unidirectional pipe connection. *Advertisements* are the means by which peers learn about network resources, such as peers, peer groups, pipes, and services. Advertisements are language-neutral meta-data structures formed as XML documents. Peers publish advertisements to announce the resources or services they provide.

We implement peers as JXTA peers. We extend the standard JXTA peer advertisement to encapsulate the schema information of a peer. Once a peer advertisement is located, a node extracts the database schema of a particular node, and matches it with its local database schema [16, 12]. The matching results show how the elements (e.g., relations, attributes) of one schema correspond to the elements of another. Its purpose is to enable the system (probably with the help of the user) to build up the acquaintance query, correspondence rules and coordination rules. Then, the two nodes exchange pipe advertisements, and create input and output pipes. At this moment, peers are said to be acquainted.

In order to implement interest groups we encode database related functionalities into a set of custom services a JXTA group will provide. We call these custom services as *DB-related services*. We need several DB-related services. We classify them into two categories: *node-level services* and *group-level services* (see Figure 1). One example of node-level service is the service which is responsible for handling of queries and query results, coming from the network. When activated, this service listens to the input pipe endpoints of known nodes. Once received a query message, the service parses it and activates query elevation procedure. If the service receives query results, then it calls a procedure for the translation, and either propagates them further or reports them to the user.

One example of group-level service is the Screening service, obtained by modifying the JXTA Membership Service. The objective of this service is to support a proper constitution of an interest group with respect to its topic. A peer willing to join a peer group, first locates a current member, and then applies for the membership, providing its schema information as credentials. The application to join is accepted or rejected by a collective set of current members. In particular,
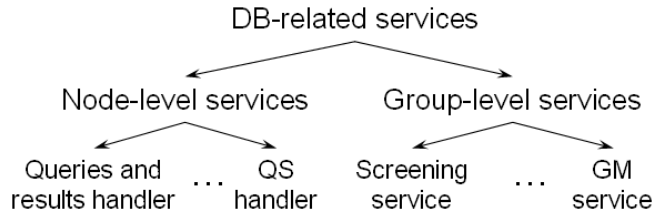
Fig. 1. Classification of DB-related services

GM checks up to what extend the local schema of the applicant can be expressed by the global schema of the group. Another group-level service provides a peer with the *GM* functionality. We encode information about the input pipe of *GM* into the set of services of a given group. Nodes use this information to contact directly *GM* for sending query scope requests, for instance. We also extend the standard JXTA peer group advertisement to include the group topic information. We build the DB-related services on top of the core services provided by JXTA. As a consequence, the implementation of the basic P2P functionality (e.g., discovery, pipes) is already given.

4 The logical architecture

We describe the logical architecture at two levels of detail: the structure of a node in a P2P database network; and the second level, which shows how the four basic notions are implemented in JXTA.

Consider Figure 2. A node consists of *P2P Layer*, a *Local Database* (LDB) and a *Database Schema* (DBS). DBS describes a shared part of LDB. The P2P Layer consists of *User Interface* (UI), *Query Manager* (QM), *JXTA Layer* and *Wrapper*. QM processes both user queries and queries coming from other nodes. It is also responsible for query results processing which come both from LDB and the network, as well as for propagating them to the network. The JXTA Layer is responsible for all node's activities on the network, such as discovering of new nodes and interest groups, joining and leaving groups, communication with group managers, sending and receiving queries and query results, and so on.

Arrows between UI and QM as well as arrows between Wrapper, JXTA Layer and QM have the same graphical notation bacause they represent procedure calls between different modules. Bidirectional arrow from JXTA Layer to a P2P database network has a different notation since it represents JXTA-supported messages, mostly represented in the form of XML documents. The arrows between DBS and LDB, and Wrapper have yet another notation since the communication they denote is LDB dependent.

In Figure 3 we open the QM and JXTA Layer. Rectangles with rounded corners stand for data repositories which store various information. Rectangles
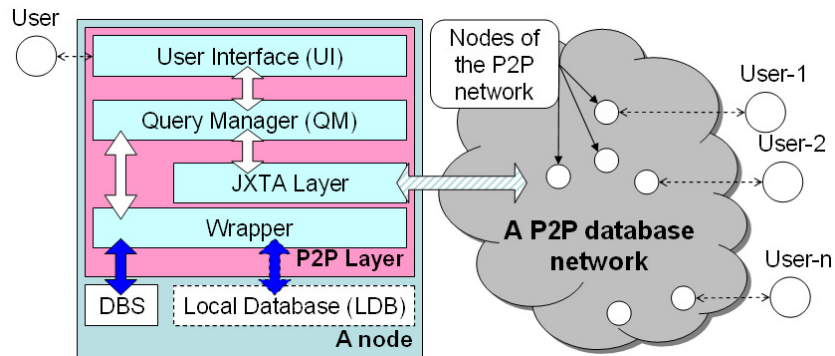
Fig. 2. First level architecture: a node

represent executive modules. The meaning of arrows between UI, QM, JXTA Layer and Wrapper is the same as in Figure 2, namely, they represent procedure calls. Continuous thin arrowed lines show information flows between modules and data repositories, as well as procedure calls between modules. Dashed arrowed lines show the dependencies between components. For example, they show that coordination rules, correspondence rules, acquaintance queries, peer advertisement and pipes all depend on acquaintances.

Consider the JXTA Layer. The advertisements repository stores all discovered and locally created JXTA advertisements. Inside the rectangle, three main advertisement types are represented, although in practice there are also others. The peer group advertisement includes also the group topic information, and the peer advertisement includes the database schema information. The Services module implements the core JXTA services and DB-related services. We encode the input pipe advertisement of the group manager in the Services module. The Discovery module implements the Discovery Service. The Pipes module implements the Pipe Service.

Consider now the Query Manager. Query Planner (QP) processes queries coming both from UI and from input pipes. For queries coming from UI, QP detects the destination groups (see link from `Peer Groups` to QP), sends QS requests to GM, and processes QSs afterwards. QP also evaluates queries w.r.t. coordination rules, and, as the result, decides where a given query should be propagated. Then, QP sends to the Query Propagation module information where to propagate a query. The latter uses the Correspondence Rules repository for query translation, and propagates queries on the network (see arrow from `Query Propagation` to `Pipes`). The Query Propagation module is also responsible for sending to Wrapper both user queries and queries coming from the network. The Results Handler receives results coming from acquaintances and translates them using Correspondence Rules. If these results are for a user query, then Results Handler reports them to UI. Otherwise, it sends them backward to the node which sent the query. Apart from this, the Results Handler gets results coming
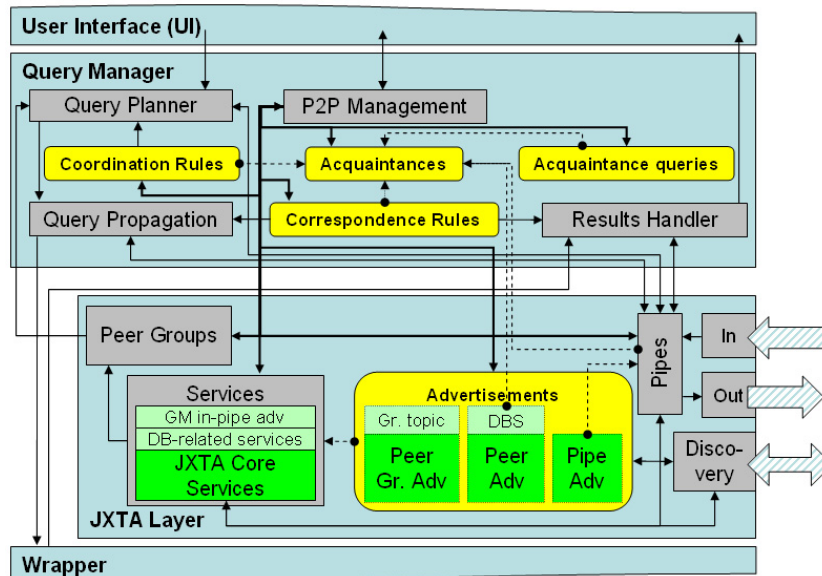
Fig. 3. Second level architecture: QM and the JXTA Layer

from Wrapper, and sends them to UI or to the network. The P2P Management module allows users to control the modules and repositories from both Query Manager and JXTA Layer. For instance, it makes it possible to create a new pipe, to make a new acquaintance or to tune up a coordination rule. The control links are shown as thick arrows from P2P Management to other components.

5 Conclusions

In this paper we have proposed a new solution which allows databases to inter-operate in a modality coherent with the P2P nature. We have defined the solution in terms of four basic notions (Interest Groups, Acquaintances, Correspondence Rules and Coordination Rules) and have shown how they allow for the implementation of database coordination. We have also shown how our solution can be implemented in JXTA and proposed a logical architecture at two levels of details.

Currently, a preliminary version of a prototype of a P2P database system is running at the University of Trento. The prototype was developed in collaboration with the Belarusian State University (www.bsu.by).

References

1. Halevy, A., Ives, Z., Suciu, D., Tatarinov, I.: Schema mediation in a peer data management system. ICDE (2003)

2. Kementsietsidis, A., Arenas, M., Miller, R.: Data mapping in peer-to-peer systems. ICDE (2003)
3. Ng, W., Ooi, B., Tan, K., Zhou, A.: Peerdb: A p2p-based system for distributed data sharing. ICDE (2003)
4. Bernstein, P., Giunchiglia, F., Kementsietsidis, A., andL. Serafini, J.M., Zaihrayeu, I.: Data management for peer-to-peercomputing: A vision. WebDB (2002)
5. Ullman, J.: Information integration using logical views. Theoretical Computer Science (1997)
6. Giunchiglia, F., Zaihrayeu, I.: Making peer databases interact - avision for an architecture supporting data coordination. 6th International Workshop on Cooperative Information Agents(CIA-2002), Madrid, Spain, September18 -20 (2002)
7. (project, J.) see http://www.jxta.org.
8. Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I.: A distributed algorithm for robust data sharing and updates in p2p database networks. Proceedings of the P2P&DB international workshop, Heraklion - Crete, Greece (2004)
9. Halevy, A.: Answering queries using views: a survey. VLDB Journal (2001)
10. Kirk, T., Levy, A.Y., Sagiv, Y., Srivastava, D.: The Information Manifold. In Knoblock, C., Levy, A., eds.: Information Gathering from Heterogeneous, Distributed Environments, Stanford University, Stanford, California (1995)
11. Hull, R.: Managing semantic heterogeneity in databases: A theoretical perspective. Bell Laboratories (1997)
12. Giunchiglia, F., Shvaiko, P.: Semantic matching. "Ontologies and Distributed Systems" workshop, IJCAI (2003)
13. Dayal, U., Hanson, E., Widom, J.: Active database systems. Modern Database Systems (1995) 434–456
14. Giunchiglia, F., Kumar, S.: A java implementation of coordination rules as ECA rules. Technical report at DIT, the University of Trento, Italy (2003)
15. Kantere, V., Kiringa, I., Mylopoulos, J., Kementsietsidis, A., Arenas, M.: Coordinating peer databases using ECA rules. DBISP2P (2003)
16. Rahm, E., Bernstein, P.A.: On matching schemas automatically. VLDB Journal 10, 4 (2001)