UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

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

MAKING PEER DATABASES INTERACT – A VISION FOR AN ARCHITECTURE SUPPORTING DATA COORDINATION

Fausto Giunchiglia and Ilya Zaihrayeu

June 2002

Technical Report # DIT-02-0012

Also: this paper will appear in the "Proceedings of the Conference on Information Agents (CIA 2002), Madrid, September 2002.

Making peer databases interact – A vision for an architecture supporting data coordination

Fausto Giunchiglia and Ilya Zaihrayeu

DIT – Dept. of Information and Communication Technology
University of Trento, 38050 Povo, Trento, Italy
{fausto, ilya}@dit.unitn.it

Abstract. Our goal in this paper is to study the problem of the interaction among databases in a peer-to-peer (P2P) network. We propose a new approach, that we call *"data (base) coordination"*, that rejects the assumption, made for instance in data integration, that the involved databases act as if they were a single (virtual) database, modeled as a global schema. From an operational point of view, the distinguishing feature of data coordination is that many of the parameters (metadata) influencing the interaction among peer databases are decided at run time. For any given query, the involved databases interact using the most "appropriate" (virtual) schema. This is crucial for dealing with the strong dynamics of a P2P network. We provide four basic architectural notions and hint how they are the building blocks of a possible distributed implementation capable of coordinating databases in a P2P network.

1. Introduction

> "Peer-to-peer is a class of applications that take advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must … have significant or total autonomy of central servers." (Quote from Clay Shirkey [20])

Among many others, the definition given above is the one providing quite a suggestive view or peer-to-peer (P2P) computing. Many examples of P2P computing already exist; take for instance Napster [16], a shared directory of available music and client software which allows, among other things, to import and export files; Gnutella [7], a decentralized group membership and search protocol, mainly used for file sharing; or Groove [9] a system which implements a secure shared space among peers. In this context, JXTA [11] is an important project which aims at creating a common platform that makes it simple and easy to build a wide range of distributed services and applications in which every device is addressable by a peer.

In such an application domain, the question which arises is whether there is a role for peer databases, also called P2P databases, by which we mean a database able to operate in a P2P environment, and therefore to interact with its peers, with a modality

coherent with the spirit of P2P computing. Very little work has been done on this problem, see for instance [8], mainly concentrating on data placement. It is still an open issue whether the development of P2P databases is simply a new problem domain where we can apply existing database technology, and in particular that developed for data integration [10], or whether the solution of this problem requires the development of new ideas, new theory, and new technology.

We believe that the development of P2P databases does require such new developments. Our goal in this paper is to argue in favor of this thesis, discuss domain characteristics and solution desiderata, and to provide the first guidelines of a possible architecture. We propose a new approach, that we call *data (base) coordination*, that rejects the assumption, made in previous approaches, most noticeably in data integration, that the involved databases act as if they were a single (virtual) database, modeled as a global schema. We talk of coordination, very much in the spirit of [15], as further elaborated in [17], meaning that ...

"... Coordination is managing dependencies between interacting databases."

From an operational point of view, the distinguishing feature of data coordination is that many of the parameters (metadata) influencing the interaction among peer databases are decided at run time. The involved databases are not integrated to implement an *a priori* defined global schema, but, for any given query, they coordinate in order to define and use the most "appropriate" (virtual) schema. This is the crucial feature which allows us to deal with the strong dynamics of a P2P network.

The paper is structured as follows. In Section 2, we articulate the database coordination problem as four related but orthogonal subproblems: database integration, database coordination, providing good enough answers, and tuning coordination over time. In Section 3 we hint at a possible architecture implementing data coordination and introduce its four basic notions, namely: interest groups, acquaintances, coordination rules, and correspondence rules. In Section 4 we provide the highlights of a query answering algorithm. In Section 5 we develop an example. We conclude with a description of some of the many open research problems (Section 6) and with some final remarks (Section 7).

Two observations. First, while we believe that the ideas described in this paper apply to both query answering and update propagation, this paper mainly focuses on the former problem. The latter problem is discussed only sporadically, and never thoroughly. Second, in this paper, after providing the basic intuitions, we concentrate on the issues concerning a possible implementation architecture. As also discussed later, our proposed architecture can be seen as an implementation of the *Local Relational Model (LRM)*, a new model and proof-theoretic framework which provides a foundations to the coordination of P2P information sources [19].

2. The database coordination problem

We articulate the database coordination problem in four steps, starting from data integration up to the issue of how to tune coordination over time.

2.1 Database integration

Let us consider the following example scenario. Consider the situation where John, a person living in Toronto, is described in the database ***F*** of his family doctor, in the database ***P*** of a pharmacy, and also in the database ***H*** of the hospital where he once received medical treatment.

In the scenario hinted above the databases are completely *autonomous* and independent of one other. They are independent in their language, contents, in how they answer queries, … They may be incomplete, overlapping, semantically heterogeneous, mutually inconsistent, … . Nevertheless it is definitely worthwhile to integrate their information and to exchange queries and updates. Consider the following examples.

Example 2.1: John is admitted to the hospital. As a consequence, ***H*** becomes "acquainted" with ***F*** for the purpose of retrieving his case history, and also for updating ***F*** with a new record corresponding to the medical checkup or aid taken when the treatment is over.

Example 2.2: The pharmacist administrating ***P*** prescribes a drug to John and, as a result, a new record is added to ***P***. A new corresponding record should appear in ***F*** as well. Therefore, ***P*** gets acquainted with ***F*** and ***F*** can be updated.

We can suppose that John always goes to the same pharmacy and hospital. In this case, once the databases get acquainted, it is possible to reuse the same data integration mechanisms for query and update exchange. If one can also predict what ***F***, ***P***, and ***H*** are, as it can be the case, then one can decide that it is worthwhile, at design time, once for all, to implement the integration among these three databases. For this kind of problems existing technology, for instance in data integration [10], may suffice.

2.2 Database Coordination

In a P2P environment things can get much more complicated. Consider the following further example.

Example 2.3: John goes to a ski resort in another country, for instance Trentino in Italy. Unluckily, here he has an accident; for instance, he breaks a leg, and he must get medical aid to the resort's medical office. This office, in turn, has its own database ***M*** which now needs to get involved. ***M*** may need to query ***H*** for the purpose of retrieving treatment details of a similar past accident. Furthermore, when John returns home, a new record from ***M*** should appear in ***F***. However the acquaintance between

M and ***F*** does not need to be maintained for ever, since the two databases will probably not need to coordinate again, and can eventually be dropped.

In situations like that described in Example 2.3 the design and development of data integration mechanisms for randomly acquainted databases which may need to communicate only a few times, becomes impractical.

We have (at least) three kinds of unpredictable run time factors, which influence the answer to a given query in a P2P network, namely:

1. *Network (dependent) variance:* the network changes over time. This may happen for many reasons, for instance: there is a different set of peers; due to their autonomy, some databases change their interaction with their peers; or, the interaction mechanisms change.
2. *Database (dependent) variance:* for any given P2P network, different databases, even if asked the same query, and at the same time, will provide different answers. While trying to provide a "global" answer, a database queries other peer databases. However each database queries the network from "its point of view". It may involve different databases (a subset of those available) or, even keeping the same set of databases, it may activate a different form of interaction with, as a result, different data passed around.
3. *Query (dependent) variance:* different queries, even if posed to the same database, will impose different points of view on the network. As above in item 2, depending of the specific query, we may cause different databases to be involved or different forms of interaction among databases.

In our opinion, network, database and query variance, are the three fundamental elements of variability which must be considered in a P2P scenario. In order to solve these kinds of problems we need new and extremely flexible mechanisms for database interaction, that we collect under the heading of *database coordination.*

Notice that, in this context, it makes very little sense to speak of a global schema, as it is commonly done in the data integration literature [10]. The main conceptual reason for rejecting this notion is that we cannot think of a set of P2P databases just as a particular implementation of a single (virtual) database, this being the underlying assumption which motivates the definition of a global schema. Consider the effects of the three kinds of variance defined above on a global schema. Network variance forces us to assume that the global schema changes over time. Database variance forces us to assume that we have a global schema for each database (access point?) in the P2P network. Query variance forces us to assume that we have a global schema for each distinct query. That is, if we make the global schema assumption, we are able to explain how any specific query posed to a specific database in a specific instant in time will be answered. However, we cannot explain how the dynamics influence the answer to a query, this being the main issue in a P2P network.

From a foundational point of view, any theory developed under the assumption of a global schema, and under the implicit assumption that the global schema is fixed, prevents us from the studying the dynamics of a P2P network. As far as we know

these two assumptions have never been relieved, in particular in the data integration literature, see for instance [10, 12]. Rejecting this hypothesis requires a new kind of semantics. It is no longer possible to see the global schema as a view of the local database (global-as-view approach) or, vice versa, the local databases as views of the global database (local-as-view approach). For instance, we can no longer assume that there is a unique universe containing all the elements of the single databases. The mappings among elements may change over time, or may be different depending on the query or on the database answering the query. This topic is not discussed in this paper. A new semantics, called the *Local Relational Model*, based on the *Local Models Semantics* [6, 4], which provides the foundations for the coordination of P2P databases is given in [19, 1].

From an implementation point of view, we believe that the previous work on data integration can be partially re-used in this framework. Once one fixes the query, the database it is posed to, and the P2P network, then it should be possible to implement coordination using some variation of the existing data integration algorithms. However, for this to happen, two steps must be taken:

1. a specific set of parameters (metadata) must be defined which can be used to model network, database and query variance;
2. the existing algorithms for data integration must be modified and parameterized as functions of the defined metadata.

2.3 Good Enough Answers

Moving from data integration to data coordination, it becomes hard to maintain a high quality level in the answers that the P2P network is able to provide. By high quality level we mean the fact that data can flow among the databases preserving (at the best possible level of approximation) soundness and completeness. In this context, soundness means that the data provided by the local databases satisfy the global schema (but they are not necessarily complete, some of them can get lost in the coordination). Completeness has the dual meaning. In the data integration literature, completeness is often given up, still maintaining the request of soundness. In a P2P environment, completeness and soundness will be very hard to achieve. This will happen in limit cases, for instance with low dynamics, simplified interaction among the databases, and if and when there will be interest in investing a substantial amount of money in the solution of the problem.

One area where there will often be interest in getting very high quality data integration is the medical care domain. There are however many other application domains where this is not the case. One such example is tourism. This domain, is not life critical, and in many cases the small dimension of a single business (hotel, campsite, …) does not justify big investments. Consider the following example.

Example 2.4: When planning his vacation in Trentino, John goes to a local agency, which unluckily can not offer John anything from their own database. Instead the

agency searches for single operators in the Trentino region (hotels, ski resorts, etc), starts communication sessions with some of them, and queries for the necessary information (e.g., prices, conditions, availability).

Compared with the medical care example, the dynamics will have a much higher impact on the quality of the answer. We have network variance: the relevant databases are much more unstable in their being active and coordinated in the network, nodes come and go (for instance depending on the season), and so on. We have database variance: John travels around and queries different databases. The same query will get different results since each database will implement different degrees of coordination with the others, and so on. Thus, for instance, a query about hotels made to a hotel database will likely get an answer that is better than the answer obtained from a campsite database. We also have query variance: if you ask a query about campsites to a campsite database you will likely get a better quality answer than if you ask this database a query about hotels, and dually for the hotel databases. Depending on the query, certain coordination mechanisms may or may not be activated. However, in this application, the agency doesn't need the best possible answer. It simply needs some answer. As long as, for instance, it gets a hotel John likes, this is good enough. Compared to the previous example, much lower quality data coordination will probably suffice.

The medical care and tourism domains are just examples. Things can get even more radical and complex when one thinks of applications where some of the nodes are mobile and where coordination happens on an even more occasional basis, for instance due to the physical proximity of two mobile peers. (As from the quote in the introduction, these kinds of situations should be quite usual in a P2P network.) In these situations, and for certain kinds of applications, almost any answer will suffice, as long as there is one. In a P2P environment, in terms of quality of answers, it is possible to go from one extreme to the other. On one extreme, it may be usual to get poor quality answers. This may happen because the databases interact partially or do not interact at all or, even worse, they pass around data which are wrong (for instance because of unsolved problems of semantic heterogeneity). On the other extreme, there will be a tight coordination and it will be possible to achieve or, at least, approximate soundness and completeness. Between these two extremes there is a continuum of answers of different quality.

The problem of the quality of the answers must be dealt with by developing the notion of *"good enough answer"*. The intuition is that an answer will be good enough when it will serve its purposes given the amount of effort made in computing it. A lot of research needs to be done on this topic. Here it is worth pointing out that, when trying to get good enough answers, one can work to produce the best infrastructure but also, for any given infrastructure, to spend a lot of time in the run time query answering. A theory of what it means to be a good answer should take into account both these factors and relate them to how the query results are used.

2.4 Tuning Coordination Over Time

In order to implement database coordination, a lot of metadata needs to be produced and maintained. Due to the strong dynamics of a P2P network, this is a crucial and hard task to perform. A node will never know the full list of its peers, it will never know everything about them, and its knowledge will be hard to maintain and will easily become obsolete. There will likely be a need of tuning and sometimes improving, on each single peer, the quality of the interaction (for instance, with the help of learning algorithms, metadata editors, and so on. See later). There is an obvious trade-off between the quality of the answers and the effort made in maintaining coordination.

3. Hints of a Possible Architecture

A P2P network consists of an open-ended number of nodes (or peers), where each peer is uniquely identified by its *id* or *address*. In our approach each peer has a local database, and an extra layer, called the *P2P layer*, or, also, the *LRM layer*. The LRM layer interacts with the local database and interfaces it with the P2P network. The resulting architecture, first published in [1], is shown in Figure 1.

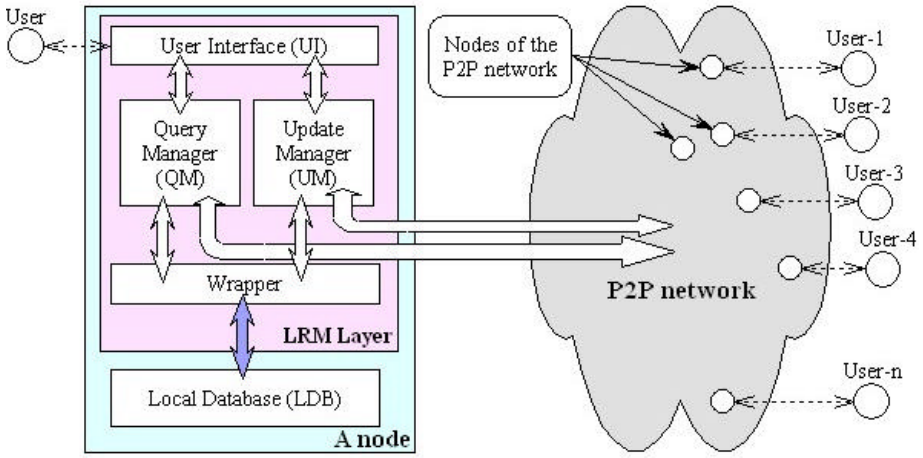

Fig. 1. First level architecture

The *LRM Layer* is the P2P functionality layered on the *Local Database* (LDB). *User Interface* allows the user to submit queries which will then be answered by the local database and the peer databases, to receive results and messages from the other nodes, and to control the other modules of the LRM Layer. *Query Manager* (QM) and

Update Manager (UM) are responsible for query and update propagation. *Wrapper* is a translation layer between QM and UM, and LDB. Peers communicate through QM and UM using XML messages (white arrows). Inter-module communication is also XML-based, as shown again by the white arrows. The communication language between Wrapper and LDB is LDB-dependent (it could be SQL, for example).

The main functionalities for coordinating P2P databases are implemented within QM and UM. These functionalities are implemented using four basic notions, described below.

Interest Groups. In most cases, nodes know very little of the other nodes of the P2P network, and in particular about the *topics* about which their peers are able to answer queries. Intuitively, medical care, tourism, tourism in Trentino, are all possible topics. A topic could be formalized as keywords, a schema, an ontology, or as a context [5], as used in [2]. We introduce the notion of (*interest) group* and define it as a set of nodes which are able to answer queries about a certain topic. A node can belong to multiple groups. The notion of group is introduced with the main goal of computing, for any given input query, the *Query Scope* (QS) – the set of nodes a query should be propagated to. The definition of a group must satisfy two complementary requirements. First, groups help deal with the complexity and the high number of nodes of a P2P network. Instead of searching for single nodes which could answer its input query, a node looks for one or more groups according to their topic. This, of course, means that the input query must be associated with a topic (this could be done by the user or by the system itself). Second, groups are used to compute a bound on the number of nodes in the query scope, therefore guaranteeing termination. Thus, topics should be general enough to capture most of the "relevant nodes", but not too general, to avoid inefficient query answering. At the moment we are supposing that each group will have a node, called the *Group Manager* (GM) which is in charge of the management of the metadata needed in order to run the group.

Acquaintances. *Acquaintances* are nodes that a node knows about and that have data that can be used to answer a specific query. A node is an acquaintance of another node only with respect to (possibly, a schematic representation of) a query. Acquaintances can therefore be thought of as links from one node to another, labelled by a (schematic) query. If a node is an acquaintance, then there must be a way to compute how to propagate a query, to propagate results back, and to reconcile them with the results coming from the other acquaintances. Crucial for these tasks are *coordination rules*, and *correspondence rules*, as defined below.

The schematic representation of queries used in acquaintances is conceptually different from the topic associated with a group. A node can be part of a group without being an acquaintance of all the nodes in that group. One example is the case where a node gets a query about hotels in Italy and chooses the group with topic "tourism in Italy". However one of the nodes in that group is a campsite. Conversely, we may have a node which is acquainted with the node that gets a user query asking about hotels, but it belongs to the group with characteristic topic "tourism in Austria".

The intuition is that a node will try to propagate a query to the acquaintances in QS. This is in turn may activate the recursive propagation of the query to the

acquaintances of the acquainted nodes. This will happen until the query is propagated to all reachable acquaintances. In general not all the nodes in QS will be queried, this happening when there is node which cannot be reached by the transitive propagation of queries along acquaintance links.

Coordination rules. Each acquaintance may be associated with one or more *coordination rules*. At run time, nodes use coordination rules which specify under what conditions, when, how and where to propagate queries or updates. One possible implementation of coordination rules is as *ECA rules* [3]. A triggering *Event* can be an update or a query coming from the user or from another node; *Condition* refers to properties of the update or query (e.g., the type of query and/or which data items are referenced by the query), and *Action* can be the translation and propagation of a given update or query to a particular acquaintance.

Consider Example 2.3 and suppose that ***M*** has a schema "Accidents". Then one possible coordination rule, relating ***M*** to ***H***, is as follows:

Event: "Query to 'M:Accidents'"
Condition: "Value of First Name attribute in the body of the query is 'John'" (**1**)
Action: "Translate that query using Correspondence Rules and send it to ***H***"

(1) is a simple coordination rule which launches query propagation to ***H*** when a query posed to ***M*** contains 'John' as a search criterion. It is easy to think of more complex coordination rules, for instance, rules which can selectively extract information from one of two databases, or rules which do some filtering, and so on.

Correspondence rules. Each acquaintance is associated with one or more *correspondence rules.* Correspondence rules take care of the semantic heterogeneity problem. They are implemented as rewrite rules and are called by coordination rules, in the body of the code implementing their action and condition components. Correspondence rules are used for the translation of queries and query results. They can be used, for instance to translate attribute or element names. In this latter case we also call them *Domain Relations* [19].

Consider the coordination rule in (1). Then we may have the following rewrite rules concerning attributes:

Address_Reason	Disease	(**2**)
Treatment_Taken	Treatment_Desc	
Prescription_Given	'None'	
Date	In	

Where "None" means that there is no corresponding attribute. Similarly, the following domain relations are applied for rewriting element names:

Value (Address_Reason)	Value (Disease)	(**3**)
Value (Treatment_Taken)	Value (Treatment_Desc)	

In the above example values are translated without modifications, but it is easy to think about more complex translations when, for instance, dates are converted to different formats, currency conversion, and so on. In practice, this is a very hard task which involves a lot of data scrubbing and transformation and which occupies a substantial amount of all the data integration projects.

4. Answering Queries

Reconsider Figure 1. The process starts when a node, let us call it *n1*, receives a user query with an indication that this is a *global query*, namely a query whose answer should be computed asking not only the local database but, also the "reachable" peers in the network. Below a list of problems which must be dealt with to answer a global query.

1. *n1* computes the query topic, maybe with the help of the user;

2. *n1* matches the query topic with the topics of the known groups and, as a result, computes one or more groups that could provide meaningful answers;

3. *n1*, with the help of QM, computes QS;

4. *n1* may not be acquainted with any of the nodes in QS, or, more in general, the acquaintances graph may not be connected enough. In order to solve this, a *Getting Acquainted Protocol* needs to be activated whose main goal is to learn coordination and correspondence rules between *n1* and the nodes in QS, or between any two nodes in QS. The getting acquainted protocol will presumably involve database schema exchange, schema matching [18], coordination and correspondence rules derivation from the results of matching, and, probably, some other phases.

5. Exploiting coordination and correspondence rules, *n1* sends the query to the acquaintances in QS. For efficiency reasons, this should be done in parallel. Here, various non trivial issues arise. In particular:

    a. Not all the nodes in QS are acquaintances of *n1*. To maximize the query results, and exploiting the fact that coordination and correspondence rules tell us how to propagate queries forwards and results backwards, the acquaintances of the acquaintances of *n1* can be *transitively* queried. This originates a propagation mechanism on the graph of acquaintances.
    b. The acquaintances graph may be a DAG, namely have multiple paths between two nodes. As a consequence, a node can be queried more than once. This requires the implementation of mechanisms for avoiding multiple answers and multiple query propagations.
    c. The acquaintances graph may have cycles. Loops must be avoided.
    d. Not all the nodes in QS may be reachable by the acquaintances graph. We have to stop for one of two reasons: we have reached all the nodes in

QS, or the nodes in QS which have not been reached yet cannot be reached.

Problem a. can be dealt with by implementing, inside each node, automated query propagation mechanisms. The implementation of coordination rules as ECA rules is one possible solution (see example above). Problems b. and c. can be dealt with by associating each query with a unique identifier. Any node receiving the same query twice will discard it. Problem d. can be dealt with by supposing that a node, for instance GM, knows QS and which nodes have been involved by the propagation mechanism (for instance because they send this information to it). GM seems a good choice as it is the node with the most information about its peer nodes, and all the nodes know about it;

6. All the nodes involved send their results back to *n1*. This is applied recursively until all the results have been propagated back to *n1*;

7. Reconcile all the results and answer the user query.

The basic algorithm hinted above can be made more efficient. Some of the possible extensions are listed below. QS can be recursively computed, for instance by the nodes answering the query; this may lead to better answers. It is possible to use intermediate nodes that are not part of QS but that, via the appropriate coordination and correspondence rule, can propagate the query; this may allow to reach otherwise unreachable nodes. Data reconciliation can be done at the intermediate nodes; among other things this can make the process faster, for instance by avoiding the propagation of duplicate information coming from different nodes. And so on.

5. A Medical Care Scenario

Let us instantiate the intuitions described above to Example 2.3 above. For simplicity we assume that all databases are relational, each consisting of one relation, and that SQL is the language used.

5.1 The three databases

Let us suppose that the relations of the databases ***F, H*** and ***M*** are as follows:

F: Prescription (PatID, P_Name, Illness_Desc, StartDate, RecoveryDate, Treatment, Type, Prescriptions);
H: Patients (PID, Name, Disease, Treatment_Desc, In, Out);
M: Accidents (P_id, FN, LN, Address_Reason, Treatment_Taken, Prescription_Given, Date);

PatID, PID and P_id are the patients' identifiers; P_Name and Name are their full names; FN and LN are their first and last names; Illness_Desc, Disease and Address_Reason are their medical problems; Treatment, Treatment_Desc and Treatment_Taken are

descriptions of the prescribed treatments; Prescriptions and Prescription_Given are the descriptions of prescribed drugs; StartDate, In and RecoveryDate, Out are the start and end dates of a treatment; Date states when certain treatment was given; and, finally, Type specifies where a treatment was conducted, a value from the set {"Home", "Hospital"}. The three databases are heterogeneous. They use different relation and attribute names to represent similar concepts, different formats for patients' ids and dates, and they also contain different data.

Let us assume that the databases keep the following items:

F:Prescription

PatID	P_Name	Illness_Desc	StartDate	RecoveryDate	Treatment	Type	Prescriptions
8	John	Flu	Mar 13, 02	Mar 15, 02	None	Home	Nasol
2	Eric O'Neill	Headache	Jan 06, 02	Jan 06, 02	None	Home	Aspirin
8	John	Leg fracture	Nov 11, 01	Dec 23, 01	Leg put in plaster	Hospital	Rest at home

H:Patients

PID	Name	Disease	Treatment_Desc	In	Out
P12	John	Abscess	Surgical operation	Mar 7, 2002	Mar 14, 2002
P10	Mary	Arm fracture	Plastering	Feb 12, 2002	Feb 17, 2002
P12	John	Forearm dislocation	Bandage	Jan 08, 2002	Jan 13, 2002

M:Accidents

P_id	FN	LN	Address_Reason	Treatment_Taken	Prescription_Given	Date
A12	Paolo	Traverso	Back injury	None	Heating ointment	01.22.02

5.2 The acquaintances graph

Let us now suppose that the following query is asked to ***M***, where 'A13' is the new id for John in ***M***:

Q_M = Select FN, LN, Address_Reason, Treatment_Taken, Prescription_Given, Date (**4**)
From "M:Accidents"
Where Address_Reason Like ('%Fracture%' Or '%Dislocation%') And PID = 'A13'

with the indication that this is a global query and that its topic is

T= "Medical care in Canada"

The intuitive meaning of query (4) is that the search should be propagated to some nodes presumably in Canada which are supposed to store information on John's case history. Note that Q_M is stated in the language of ***M***.

Let us further suppose that, after some search (possibly guided by the user), **T** is matched with the topic "Medical care in Toronto" of the interest group ***G*** = {***F***, ***H***, ***P***}, with ***H*** being ***G***'s GM. Notice that ***M*** is not part of ***G***. However this is irrelevant to our purposes. We can further suppose that ***H*** is acquainted with ***F*** and that ***P*** is acquainted with ***F***. This situation is reported in Figure 2 below.

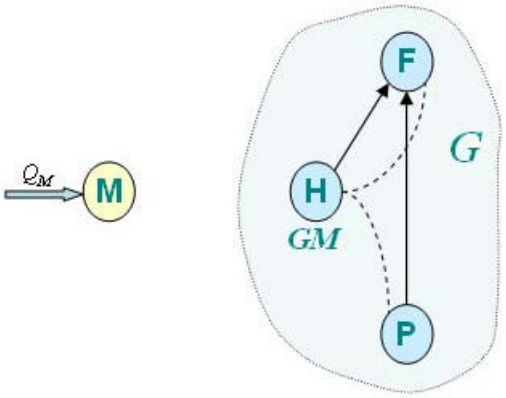

Fig. 2. Initial state

The dashed lines between ***H*** and ***F***, ***H*** and ***P*** stand for the network connections used for the propagation of the metadata needed in order to manage the group and its services. As from Section 4, these connections can, for instance, be used to propagate to ***H*** all the data needed in order to guarantee termination for any set of nodes QS which is a subset of ***G***.

Let us suppose that GM computes and sends back to ***M*** a query scope QS = ***G*** = {***F***, ***H***, ***P***}. To start the coordination session, ***M*** must get acquainted with one of the nodes in ***G***. We can reasonably assume that it is decided that ***M*** must get acquainted with ***H***.

At this point the *Getting Acquainted Protocol* is activated. The schemas of M:accidents and H:patients are matched. They both have the structure of a depth 1 tree. The two structures can be easily matched once the elements are matched, for instance by applying, among others, linguistic techniques.

As a result a set of coordination rules are learned. They are:

COOR#1 **(5)**

Event: M:Q

Condition: Q: (Address_Reason ∈ Select OR Treatment_Taken ∈ Select)
AND (PID = 'A13' ∈ Where)

Action: Q = Apply (Q, Corr#1);
Q = Apply (Q, Corr#2);
Q = Apply (Q, Corr#3);
Q = Apply (Q, Corr#4);

Q = Apply (Q, Corr#5);
Q = Apply (Q, Corr#6);
Q = Apply (Q, Corr#7);
Q = Delete_not_Mapped (Q);
Send (Q, H).

COOR#1 is the rule identifier. **COOR#1** propagates the input query to ***H*.** It triggers when a query Q is submitted to ***M***. In the condition part we check whether the select part of Q refers to Address_Reason or Treatment_Taken (these attributes are shared with ***H***), and whether the patient id is used in the query condition part and it is equal to John's id (a possible shared patient between ***M*** and ***H***). The action part of the coordination rule defines the sequence of query modification operations, and it concludes by sending the modified query to ***H***. Apply (Q, Corr_rule) returns Q after applying the correspondence rule Corr_rule. Delete_not_Mapped (Q) eliminates those attributes which are not referred by any of the correspondence rules.

COOR#2 (**6**)

Event: $M:R_H$

Condition: None

Action: R_M = Null;
R_M = Apply (R_H, Corr#8);
R_M = Apply (R_H, Corr#9);
R_M = Apply (R_H, Corr#10);

COOR#2 propagates results back from ***H*** to ***M***. There is no need to apply domain relations. **COOR#2** calls correspondence rules which indicate which value should be assigned to which attribute in ***M***.

A set of correspondence rules are also learned. Below is a set of rules for translating attribute names in a query:

Corr#1:	P_id	PID	(**7**)
Corr#2:	FN	Name	
Corr#3:	LN	Name	
Corr#4:	Address_Reason	Disease	
Corr#5:	Treatment_Taken	Treatment_Desc	

The following translate object and schema names:

Corr#6:	'M:Accidents'	'H:Patients'	(**8**)
Corr#7:	'A13'	'P12'	

Note that the matching between ***H*** and ***M*** is not perfect, and, in particular, that the attribute date is not translated. We expect that the getting acquainted protocol will not be able to produce the best possible match in many cases.

Finally, a set of rules needed for the translation of results are also learned. Note again that there is also no mapping of date related attributes.

Corr#8:	Name	FN	(**9**)
Corr#9:	Disease	Address_Reason	
Corr#10:	Treatment_Desc	Treatment_Taken	

5.3 Query propagation

Finally, when an "appropriate" acquaintances graph has been built, the propagation algorithm can be started. The resulting data flow is given in Figure 3 below.

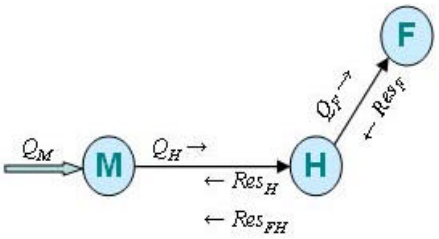

Fig. 3. Query answering

According to the rules given above Q_M gets translated to Q_H as follows:

Q_H = Select Name, Disease, Treatment_Desc (**10**)
From "H:Patients"
Where Disease Like ('%Fracture%' Or '%Dislocation%') And PID = 'P12'

H computes the following answer to Q_H

Res_H = <'John', 'Forearm dislocation', 'Bandage'> (**11**)

which is then sent unmodified to ***M*** (even if at ***M*** the same values are related to different attributes).

Following the transitivity mechanism described in $ection 4, ***H*** translates Q_H and propagates it to ***F*** as Q_F. Since ***H*** and ***F*** are from the same group we can suppose that we have correct and complete coordination and correspondence rules. Let us suppose that Q_H is translated to Q_F as follows:

Q_F = Select P_Name, Illness_Desc, Treatment (**12**)
From "F:Prescriptions"
Where Illness_Desc Like ('%Fracture%' Or '%Dislocation%') And PID = '8'

F returns to ***H*** the following query answer:

Res_F = <'John', 'Leg fracture', 'Leg put in plaster'> (**13**)

These results are translated at ***H***, propagated back to ***M***, where all the results are reconciled and presented as:

Res_H^M = <'John', 'Forearm dislocation', 'Bandage'> (**14**)

Res_{FH}^{M} = <'John', 'Leg fracture', 'Leg put in plaster'>

5.4 Variance and good enough answers

Consider the results in (14). They are incomplete and some fields (LN and Date) present in Q_M are missing in the answer. Nevertheless, these results are good enough since they still serve the needs of ***M***. LN and Date are not critically important for ***M*** for treatment purposes, moreover John can likely provide approximate dates.

Let us now consider an example of network variance. Suppose that ***F*** is down and cannot answer queries. The results produced are even more incomplete due to the fact that Res_{FH}^{M} is not present. Whether they are good enough, it depends on what it will be possible to do with only Res_{H}^{M}. Since, in Example 2.3 John breaks a leg, much it will depend on whether it is the same leg as in Res_{H}^{M}. John will be able to produce this information.

Let us now consider an example of database variance. Recall Figure 3 and suppose that ***M*** gets acquainted with ***F*** (instead of ***H***). The database variance is caused by the change in the acquaintance path (***F***, instead of ***H***, is queried). ***M*** poses, in the language of ***F***, a query Q_F which requests the same data as Q_H. However, the answer is different since ***F*** has a different 'vision' of the world. More concretely, it is not acquainted with ***H***. We have:

Res_{F}^{M} = <'John', 'Leg fracture', 'Leg put in plaster'> (**15**)

Very likely this answer will be good enough.

Let us conclude with an example of query variance. Let us suppose that we ask Q_M as above with "John" substituted with "Mary". Mary is not a shared patient between ***H*** and ***F*** and, therefore, there are no coordination rules for her and there is no query propagation to ***F***. Notice that a value in an attribute changes the query propagation tree. We have the following answer:

Res_{F}^{M} = <'Mary', 'Arm fracture', 'plastering'> (**16**)

This will likely be a good enough answer.

6. Research Problems

The ideas described in this paper are very preliminary. A lot of work needs to be done to make these ideas concrete and to be able to evaluate their usefulness. In this section we articulate, at the current state of the art, some of the open and relevant research problems. We do not consider the many issues which need to be dealt with at the theoretical level, and in particular, what we consider crucial future developments of the LRM. For a preliminary discussion about this, see [19].

Groups. A lot of issues need to be dealt with. Some of them are as follows: we need to verify the role of the group manager, to define what a topic is, and to define the services associated to a group, we need techniques for group discovery, for

associating a node to a group, for maintaining the group metadata, for propagating (part of) them in the P2P network and/or to the member nodes, …, and so on. At the moment we are evaluating whether the JXTA group mechanisms can be reused, at least in part, to implement our own interest groups.

Query answering and update propagation. This is still open space. As from above, an interesting issue is how much of the existing technology in data integration, and in particular of the LAV/GAV technology [10], we will be able to reuse and to adapt to the P2P problem domain. Similarly, concerning the implementation of coordination rules, we should be able to leverage the existing work on active databases and ECA rules, see for example [3].

Maintaining coordination metadata. We foresee two basic mechanisms for maintaining and developing metadata. The first is by (semi) automatically learning them from navigating the network. We will need to develop sophisticated *matching* techniques capable of computing the most appropriate group topic and the query scope, and to learn coordination and correspondence rules. Most of the time the match will be only partial and not perfect. The general area of matching is definitely not mature. However a lot of material can be found in the literature, see for instance [18, 13, 14]. The second is the development of (graphic) editors which should allow a user to develop, in a computer supported way, the desired metadata. To this extent, it is important to notice that coordination and correspondence rules are the procedural implementation of *coordination formulas,* as defined in [19]. Coordination formulas are indexed first order formulas and are the declarative specification of coordination and correspondence rules. Also topics, if we take them to be labeled graphs, can be developed using an editor similar to that described in [2].

7. Conclusion

This paper is a first investigation of the problem or how to make databases interact in a P2P network. P2P networks are characterized by high dynamics and by the fact that nodes are autonomous. We have characterized the problem as having four main dimensions:

1. We must integrate data coming from autonomous, most often semantically heterogeneous databases. This problem is very similar to the data integration problem widely studied in the literature;
2. We must deal with network, database, and query variance. This requires us to define a new set of metadata and to define algorithms whose behavior changes in dependence on these data. This is why we talk of data coordination, as distinct from data integration;
3. We will almost never get correct and complete answers. We must be content with answers which are good enough;
4. There is a need to tune metadata. This is requires in order to cope with the dynamics of a P2P network.

We have provided the guidelines of an architecture which supports coordination among peer databases. This architecture is based on four basic notions: interest groups, acquaintances, coordination and correspondence rules. We have listed what we consider to be the most relevant research problems in this domain. The theory underlying and motivating the architecture and notions discussed in this paper is described in [19].

8. Acknowledgements

The ideas described in this paper have matured thanks to the close interaction with Phil Bernstein and John Mylopoulos. Phil Bernstein asked a lot of hard questions. John Mylopoulos suggested that coordination formulas could be implemented as ECA rules. We also thank Paolo Bouquet, Gaby Kuper, Matteo Bonifacio and Luciano Serafini for the many stimulating discussions.

9. References

1. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I. "Data Management for Peer-to-Peer Computing: A Vision". WebDB02 – Fifth International Workshop on the Web and Databases, 2002.
2. Bouquet, P., Dona', A. and Serafini, L. "ConTeXtualized local ontology specification via CTXML". MeaN-02 – AAAI Workshop on Meaning Negotiation. Edmonton, Alberta, Canada, 2002.
3. Dayal U., Hanson E.N., and Widom J. "Active Database Systems. Modern Database Systems: The Object Model, Interoperability, and Beyond". W. Kim, (Ed), Addison-Wesley, Reading, Massachusetts, 1994.
4. Ghidini, C., and Serafini, L. "Distributed First Order Logics". Frontiers of Combining Systems 2 (Papers presented at FroCoS'98). Dov M. Gabbay and Maarten de Rijke, (Eds), Research Studies Press/Wiley. ISBN 0863802524 (hardback).
5. Giunchiglia, F. "Contextual reasoning". Epistemologia, special issue on "I Linguaggi e le macchine". Vol. XVI, pages 45-364, Tilgher-Genova, Italy, 1993.
6. Giunchiglia, F., and Ghidini, C. "Local Models Semantics, or Contextual Reasoning = Locality + Compatibility". KR'98 –Sixth International Conference on Principles of Knowledge Representation and Reasoning. Morgan-Kauffman, 1998. Long version: Ghidini, C., and Giunchiglia, F. "Local Models Semantics, or Contextual Reasoning = Locality + Compatibility". Artificial Intelligence. 127(3):221-259, 2001.
7. Gnutella, see http://www.gnutellanews.com .
8. Gribble S., Halevy A., Ives Z., Rodrig M., and Suiu D. "What can Databases do for Peer-to-Peer?". WebDB01 –Fourth International Workshop on the Web and Databases, 2001.
9. Groove, see http://www.groove.net .
10. Halevy, A. "Answering queries using views: a survey". VLDB Journal 2001.

11. JXTA, see www.jxta.org.
12. Lenzerini, M. “Data Integration: A Theoretical Perspective”. PODS 2002: 233-246, 2002.
13. Madhavan J., Bernstein P. A., and Rahm E. "Generic Schema Matching Using Cupid". Proc. VLDB 2001, 2001.
14. Magnini, B., Serafini, L. and Speranza, M. “Linguistic Based Matching of Local Ontologies”. MeaN-02 –AAAI Workshop on Meaning Negotiation. Edmonton, Alberta, Canada, 2002
15. Malone, W.T., and Crowston, K. “The Interdisciplinary Study of Coordination”. ACM Computing Surveys, Vol. 26, No. 1, 1994.
16. Napster, see http://www.napster.com.
17. Perini, A., Susi, A., and Giunchiglia, F. “Designing Coordination among Human and Software Agents”. SEKE'02 –Fourteenth International Conference on Software Engineering and Knowledge Engineering, 2002.
18. Rahm E., and Bernstein P. A. “A survey of approaches to automatic schema matching”. The VLDB Journal 10: 334–350, 2001.
19. Serafini, L., Giunchiglia, F., Mylopoulos, J., and Bernstein, P.A. “The Local Relational model: Model and Proof Theory”. IRST Technical Report 0112-23, Istituto Trentino di Cultura, December 2001.
20. Shirkey, C., see http://www.shirky.com/#p2p .