Coordinating Peer Databases Using ECA Rules

Vasiliki Kantere[1], **Iluju Kiringa**[2], **John Mylopoulos**[1],
Anastasios Kementsietsidis[1], and **Marcelo Arenas**[1]

[1] Dept. of Computer Science, University of Toronto
[2] School of Inf. Technology and Engineering, University of Ottawa

Abstract. Peer databases are stand-alone, independently developed databases that are linked to each other through acquaintances. They each contain local data, a set of mapping tables and expressions, and a set of ECA rules that are used to exchange data among them. The set of acquaintances and peers constitutes a dynamic peer-to-peer network in which acquaintances are continuously established and abolished. We present techniques for specifying data exchange policies on-the-fly based on constraints imposed on the way in which peers exchange and share data. We realize the on-the-fly specification of data exchange policies by building coordination ECA rules at acquaintance time. Finally, we describe mechanisms related to establishing and abolishing acquaintances by means of examples. Specifically, we consider syntactical constructs and executional semantics of establishing and abolishing acquaintances.

1 Introduction

Peer-to-Peer (P2P) networking is establishing itself as an architecture of choice for a certain number of popular applications such as file sharing, distributed processing, and instant messaging. A P2P architecture involves nodes, called *peers*, which act as both clients and servers, and participate in a common network by autonomously controlling and evolving its topology. Using appropriate protocols, new peers may join or leave the network at will.

Building on the P2P model of computing, data management techniques are now being developed for query, update, and transaction processing in a P2P network involving peers that are databases [3, 5]. The architecture of such a P2P network consists of a collection of *peer databases* that are located at nodes of the P2P network. Each peer database is managed by a peer data management system (PDBMS). Typically, a PDBMS has a P2P layer that interoperates the peer database with other peers. In order for such an interoperation to take place, an acquaintance must be established between the peer databases. A peer database establishes at least one *acquaintance* with other peers that are already part of the P2P network. Doing so, it joins the P2P network. The acquainted peers are called *acquaintees*. Acquaintees share and coordinate data over their acquaintance. Acquaintances are transient since they may evolve over time as peers join and leave the network. An acquaintance is established between peers that belong to the same *interest group*. The later is a community of peers centred around a common business (e.g. airline, genomic, hospital, and university databases) [4].

We assume that the peer databases do not share any global database schema and administrative authority. Moreover, the availability of their data is strictly local.

Syntactically, the peer language includes the usual constructs for querying and updating data. It also includes constructs that allow the user to establish or abolish acquaintances between peers. Semantically, an acquaintance outgoing from a peer is qualified by a subset of the mapping tables and expressions, as well as by a set of coordination rules. The former constrain the data exchange over the acquaintance, and the latter coordinate that exchange.

In this paper, we describe algorithms used by PDBMSs for establishing and abolishing acquaintances. We give several examples of coordination rules expressed in an extension of the standard SQL3 triggers. We also consider syntactical constructs used to establish or abolish acquaintances between peers. Finally, we give execution semantics for these constructs. Specifically, we make the following contributions:

- Using the concept of mapping tables introduced in [7], we present new techniques for specifying data exchange policies on-the-fly based on constraints on the way in which peers exchange and share data. Unlike many approaches used in data integration systems [9], we do not consider design-time, uniform access to a collection of heterogeneous data. Rather we consider a setting in which data coordination plays a key role. Here, each PDBMS defines and manages its own view of the shared data, and defines its own sharing and coordination policies based on Event-Condition-Action (ECA) rules provided by interest groups.
- We realize the on-the-fly specification of data exchange policies by selecting coordination ECA rules at acquaintance time from domain-specific rule libraries, rather than having them designed for particular peers by database designers. This way of selecting active rules contributes to a vision of a P2P database technology that can be used by end-users to establish acquaintances between peers in a flexible and cost-effective way.
- We focus on setting up and abolishing acquaintances between PDBMSs. We describe mechanisms to that end and illustrate them by means of examples. These mechanisms create logical meta-data and ECA rules on-the-fly to guide the exchange of data.

This work is part of the Hyperion project conducted at the Universities of Toronto and Ottawa [1].

This paper is organized as follows. Section 2 motivates the use of ECA rules for coordinating peer databases; it also reviews the notions of mapping tables and expressions. Section 3 summarizes the architecture of Hyperion. Mechanisms used by peer databases for joining and leaving a peer network are described in Section 4. Here, we show how ECA rules are generated on-the-fly and from a library of generic rules for setting up coordination between peers. Section 5 treats related work. Finally, Section 6 concludes the paper and indicates how the framework presented here is being extended in various ways.

OA_Passenger

pid	name
1	Lachance
2	Smith
3	Moore

OA_Ticket

pid	fno	meal
1	OA229	Veal
2	OA378	Trout

OA_Flight

fno	date	dest	sold	cap
OA229	01/05	Mont.	120	256
OA341	01/15	Tor.	160	160
OA378	01/21	S.F.	90	124

(a) Ontario-Air Database Instance

QA_Passenger

pid	name
1	Michel
2	Lachance

QA_Fleet

aid	type	capacity
B-1	Bombard. 27	140
B-2	Embraer L12	130
B-3	Boeing 737	117

QA_Flight

fno	date	to	sold	aid
QA2132	01/07	Dorv.	67	B-3
QA1187	01/15	LBP	118	B-2
QA1109	01/15	MDC	164	B-1

QA_Reserve

pid	fno
1	QA2132
2	QA1187
2	QA1109

(b) Quebec-Air Database Instance

Fig. 1. Instances for the two airline databases

2 ECA Rules for Coordinating Peer Databases

2.1 Motivating Example: Mapping Tables and Expressions

We consider two airline-ticket reservation peer databases *Ontario−Air_DB* and *Quebec − Air_DB* belonging to two fictitious airlines Ontario Air and Quebec Air, respectively. Assume that these databases manage reservations for flights originating in Ottawa. Their schemas are as follows:

OA_Passenger	(pid, name)
OA_Flight	(fno, date, dest, sold, cap)
OA_Ticket	(pid, fno, meal)

(a) Schema for the Ontario-Air database

QA_Fleet	(aid, type, capacity)
QA_Passenger	(pid, name)
QA_Flight	(fno, date, to, sold, aid)
QA_Reserve	(pid, fno)

(b) Schema for the Quebec-Air database

Ontario-Air stores identifications and names for each passenger. It stores flight numbers, dates, destinations, number of tickets sold, and capacities of flights. Finally, it stores identifiers, flight numbers, and meal requests of passengers. Quebec-Air stores passenger identifiers and names, and the number, date, destination airport code, number of tickets sold and the airplane identifier for each flight. Quebec-Air also stores a table containing the passenger identifiers and the corresponding flight numbers. Finally, Quebec-Air stores information about its

fleet, i.e., the identifier of each plane together with the corresponding type and capacity. Figure 1 shows instances of the two peer database schemas.

Ontario-Air and Quebec-Air can aim at coordinating their activities by establishing an acquaintance. The goal of such an acquaintance can be to favour an exchange of flight information and to coordinate their flights to various locations. Since the schemas of both peer databases are heterogeneous, no data exchange can take place before a form of homogenization is undertaken. In the more traditional context of data integration (see e.g. [9]), views are used to facilitate some form of data exchange across heterogeneous schemas. Usually, to adopt our terminology, these views constraint the *content* of source peer databases in order to determine the content of target peer databases. However, two major assumptions of the PDBMS setting precludes the use of the data integration approach: peer databases are supposed to be autonomous with respect to their contents, and the very large number of involved peers makes any static approach to the reconciliation process of peer contents inadequate. Rather we focus on techniques for coordinating data exchange between peers by looking for ways of constraining, not the contents of the peers, but the data exchange itself. We propose using mapping tables [7] and expressions [1],and an appropriate extension of SQL3 triggers as a way of constraining data exchange across heterogeneous boundaries. Figure 2(a) shows an example of the former construct, and Figure 2(b)-(c) shows examples of the latter.

Intuitively, mapping tables are binary tables that provide a correspondence between data values of acquaintees. Implicitly, they also provide a rudimentary schema-level correspondence. As an example, the mapping table 2(b) associates city names mentioned in the Ontario-Air database to airport codes mentioned by the Quebec-Air database, and the table 2(c) gives correspondences between flight numbers mentioned by the two peer databases.

Intuitively, a mapping expression relates two different schemas. It realizes a schema level correspondence of two different peers. Syntactically, it is expressed in some first order, Datalog-like language. The mapping expression of Figure 2(a) says that, in the context of data exchange, any flight of Quebec-Air is considered a flight of Ontario-Air, and not necessarily vice-versa. Here, we assume that the mapping expression is created at the Ontario-Air database.

Once the two databases are acquainted, and mapping tables or expressions are in place, the peers can use each other's contents during query answering and data coordination. In [1], the use of mapping tables in query answering is outlined and illustrated.

2.2 ECA Rules for PDBMSs

The Language: Apart from querying, peers are also able to coordinate their data with those of their acquaintances. For some applications, run-time reconciliation will not be sufficient and we will want to reconcile the data as it is updated. In this example, suppose the two partner airlines wish to reconcile their data to conform to the mapping expression of Figure 2(a). Using this mapping expression, we can derive a rule to ensure the two databases stay consistent as

$OA_Flight(fno,date,dest,sold,cap) \supseteq QA_Flight(fno,date,dest,sold,cap)$

Mapping expression 2(a)

dest	to
Tor.	LBP
Tor.	ILD
Mont.	Dorv.
Mont.	Mira.
Ottawa	MDC

OA.fno	QA.fno
OA229	QA2132
OA341	QA1187
OA341	QA1108

Mapping table 2(b) Mapping table 2(c)

Fig. 2. Mapping tables and expressions

new passengers are entered into the Quebec-Air peer. To enforce this rule, and other similar business rules, we propose a mechanism which uses event-condition-action (ECA) rules with the distinctive characteristic that events, conditions and actions in rules refer to multiple peers. An example of such an ECA rule is given below.

create trigger passengerInsertion
after **insert on** *QA_Passenger*
referencing new as *NewPass*
for each row
begin
insert into *OA_Passenger* **values** *NewPass*
in *Ontario-Air_DB*;
end

According to this rule, an event that is detected in the Quebec-Air database causes an action to be executed in the Ontario-Air database. Specifically, each passenger insertion in the QA_Passenger relation generates an event which triggers the rule above. The rule has no condition while its action causes the insertion of an identical passenger tuple in the OA_Passenger relation.

The rule above is expressed in a language extending SQL3 triggers [8]. An SQL3 trigger is an ECA rule that bears an explicit name. Each trigger is associated with a specific table, and only insertions, deletions, and updates on this table can activate the associated trigger. SQL3 events are simple. Conditions are SQL queries, and actions are SQL statements that will be executed on the database. There are two sorts of triggers, namely the AFTER and BEFORE triggers. The former are activated before the occurrence of their activating event, and the later are activated after their activating event has occurred.

A rule of our language extends a typical SQL3 trigger by explicitly mentioning the database in which the event, condition, and action occurs. It also contains

a richer event language to permit the expression of complex active behavior encountered in peer coordination. The rule above is a example of a typical rule for enforcing the consistency of peers that coordinate their work. In this paper, we illustrate the rule language and its use in coordinating data exchange between different peers. We also outline its execution semantics below. We leave its formal definition out of the scope of this paper.

Interaction with Mapping Tables: The mappings in the mapping tables are intimately related to values that are stored in the Ontario-Air and Quebec-Air databases. Therefore, it is natural to keep in the mapping tables only the associations between values that exist in the actual instances of the relation schemas of those databases. This means that insertions and deletions of values into the instances of Ontario-Air and Quebec-Air databases may result in an update of mapping tables to avoid discrepancies between the databases of both peers and the mapping tables relating them.

Using our running example, the following discussion further illustrates the updating of mapping tables. Whenever there is a deletion of a tuple in OA_Flight, there is an ECA rule that removes the appropriate tuple in the mapping table of Figure 2(c). Of course such a rule may not be needed for many cases of mapping tables. For example, as far as Figure 2(b) is concerned, even if there is a deletion of all flights of Ontario-Air to New-York, we wouldn't like to delete the respective information from Figure 2(b), because we will probably need it in the future. However, in the case of Figure 2(c), if a cancelled Ontario-Air flight becomes active, again it may not correspond to the same flight of Quebec-Air database. Thus, deleting the corresponding tuples from Figure 2(c) when a flight is cancelled is a reasonable action. The following rule illustrates how deletion in the mapping table of Figure 2(c) might be dealt with:

```
create trigger flightDeletion
after    delete on OA_Flight
         referencing old as OldFlight
for each row
begin
         delete from MT2c where MT2c.fnoOA = OldFlight.fno
end
```

Here, $MT2c$ is the relation name of the mapping table of Figure 2(c).

3 System Architecture

This section also summarizes the architecture of Hyperion [1]. Figure 3 represents a generic architecture for the envisioned data management framework. The architecture consists of a collection of PDBMSs located at nodes $N_1, \cdots, N_m$ of a P2P network. The various acquaintances that exist in the network induce the existence of communities of peers that coordinate their data. Such groups are

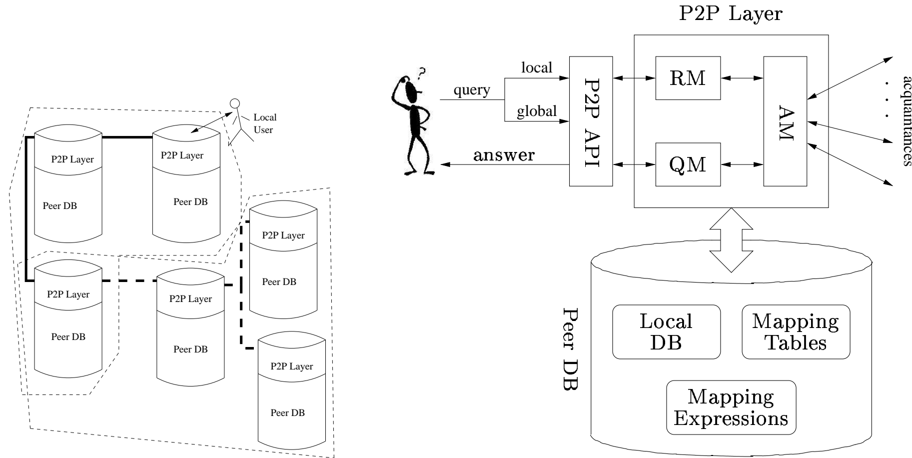

Fig. 3. Architecture for a PDBMS

called *interest groups* [4]. On the left hand side of Figure 3, two such groups, delimited by thin dashed lines, are depicted. Each PDBMS coordinates many of its typical database activities such as queries, updates, and transactions with its acquaintees in a transparent way. Our proposal assumes concepts such as absence of any global database schema, transient PDBMSs, distributed (or no) administrative authority, locally available data, strict local access to a single database, etc.

Figure 3, right hand side (taken from [1]), depicts the architecture of a PDBMS with its main functionalities. A PDBMS consists of three main components: an interface (P2P API), a P2P layer, and a DBMS. For simplicity, we do not show the local management system layer that manages a peer database. This contains local data along with mapping tables and expressions that are used in data exchange with other peers. The P2P API is the interface for posing queries and specifying whether these are to be executed only locally or remotely. Through an acquaintance manager, the P2P layer allows a PDBMS to establish or abolish an acquaintance (semi-)automatically at runtime, thereby inducing a *logical* peer-to-peer network. More specifically, the acquaintance manager uses mapping tables as constraint on the data exchange between peer databases to automatically check the consistency of a set of mapping constraints and infer new ones. In [7], these capabilities are shown to be of practical importance in establishing new acquaintances. We will return to this issue below.

4 Setting up and Abolishing Acquaintances

In this section, we describe the algorithms used by peer databases for joining and leaving a peer network. Syntactically, the peer language includes constructs that

allow the user to establish or abolish acquaintances between peers. Semantically, an acquaintance is a kind of bookkeeping instance that interface two peers. Figure 4 depicts the constraints of an acquaintance.

4.1 Setting up Acquaintances

To join a network, a peer N_i, must establish an acquaintance with a known peer, say N_j, which is already part of the network. We assume, for simplicity, that acquaintances are explicitly established by a user, most probably a database administrator. To that end, the peer language includes the following construct in its syntax:

```
set acquaintance to <peer database>
   [using mapping tables <list of mapping table names>]
   [using mapping expressions <list of mapping expression names>]
   [belonging to <Interest Group>]
```

Using this construct, a peer database administrator establishes explicit acquaintances between her database and other existing ones. When an acquaintance is established, it is coupled with several constraints. The most important of these constraints are mapping tables and expressions which – as stated earlier – constrain the exchange of data between peers, and coordination rules (written in the ECA language outlined above) which are guidelines for such an exchange.

Now we give details on the algorithm used by a peer to establish an acquaintance. Assume that peer N_i issues the following command for establishing an acquaintance:

set acquaintance to N_j

Then the following algorithm is used for completing this request:
Phase 1. Semi-automatically generate mappings as follows:

1. Use a matching algorithm to get an initial match between schemas of the peers N_i and N_j. Such a matching will most probably not be correct or complete, and may have to be revised manually by the administrator.
2. Create mapping expressions and views.
3. Use the match obtained in step 1 to create and populate an initial set of mapping tables.
4. Send a copy of the mapping table instances to N_j.

Phase 2. Generate consistency-enforcing rules from mapping expressions obtained in Phase 1,and generate rules for maintenance of mapping tables.
Phase 3. Add N_j to the list of N_i's acquaintees.

The algorithm above does not use initial mapping tables. To process a request with an initial list $M_1, \cdots, M_n$ of mapping tables, it suffices to replace step 3 of phase 1 in the algorithm above by the mere use of the given mappings tables. The algorithm above also does not use initial coordination rules. We could assume that an initial list of coordination rules is used in setting up an acquaintance.

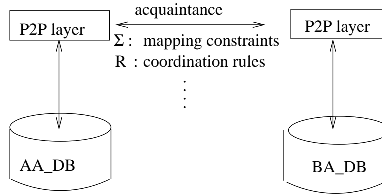

Fig. 4. A constrained acquaintance

However, though this is not precluded in principle, we prefer to think of such a list of initial rules being brought in on-the-fly from an interest group, rather than being designed for particular peers by database designers. The main reason for doing this is that we want to create a technology that is end-user oriented.

4.2 Generic Rules for Interest Groups

The simple acquaintance algorithm given above assumes that all peers belong to a single universal group of peers. However, it seems appropriate to classify peers into interest groups. We assume that an interest group has standard schemas known to all members of the group. For the airline domain, for example, we have schema S_A for airline peer databases, schema S_{TA} for travel agencies, schema S_{RA} for regional airlines. Rules are written for common patterns of data exchange and coordination among S_A, S_{TA}, and S_{RA}. When a particular set of peers, say airline a_1, travel agency ta_1 and regional airline ra_1 decide to coordinate with a given rule R_1, they need to bind/match their respective schemas to the schema with respect to which R_1 was defined in order to find appropriate mappings. With these bindings in place, all that is left is agreement among a_1, ta_1, and ra_1 that they indeed want to coordinate with rule R_1. This detail is left out.

When peers join the network, they need to register as belonging to certain interest groups. By default, the interest group will be a distinguished catch all interest group to which all peers belong. It also means that some domain experts use the rule language outlined in Section 2.1 to define possible coordinations among members of different interest groups. We assume that *generic rules* are created with respect to the standard schema of an interest group, and then adapted for specific databases that belong to the interest group. In fact, a generic rule may involve the standard schemas of more than one interest group. For simplicity, we also assume that the standard schema and generic rules are stored at every member of the interest group.

By means of an example, we now illustrate step by step a mechanism for setting up acquaintances in the presence of interest groups. In the example below, we will use the following terminology. A schema mapping is a correspondence between schema elements of two peers. We call a mapping "strict" if it represents an identity function that maps each value of an attribute A of the first peer to itself in the second peer. For example, the values of the attribute "date" in the

Ontario-Air database are mapped to itself in the Quebec-Air database. We call a mapping "loose" if it represents any function that possibly maps different values of the same attribute. Finally, a view is conceived in its traditional meaning.

Assume the schemas of Ontario-Air and Quebec-Air given in Section 2.1 and that both peers belongs to the interest group *Airlines*.

Step 1. Suppose that the interest group *Airlines* has the following standard schema, called S_A:

Ticket(pid, name, fno, meal)
Flight(fno, date, destination)
FlightInfo(fno, sold, cap, aid, type)

Then, the schema mappings for Ontario-Air and Quebec-Air databases would be as follows:

For Ontario-Air, we have the following mappings $M_{S_A \to OA}$:

M_{OA_1}: OA_Passenger(pid, name) $\leftarrow$ Ticket(pid, name, fno, meal)
M_{OA_2}: OA_Ticket(pid, fno, meal) $\leftarrow$ Ticket(pid, name, fno, meal)
M_{OA_3}: OA_Flight(fno, date, dest, sold, cap) $\leftarrow$ Flight(fno, date, dest),
FlightInfo(fno, sold, cap, aid, type)

From the three mappings above, Ontario-Air marks M_{OA_2} as a loose one (because the database administrator thinks that the mapping of fnos may not be very accurate, because she knows that usually airlines use different flight numbers). Also, M_{OA_3} is marked as a loose one (because the administrator suspects that different airlines consider different things as destinations). Finally, M_{OA_1} is marked as a strict mapping.

For Quebec-Air, we have the following mappings $M_{S_A \to QA}$:

M_{QA_1}: QA_Passenger(pid, name) $\leftarrow$ Ticket(pid, name, fno, meal)
M_{QA_2}: QA_Reserve(pid, fno) $\leftarrow$ Ticket(pid, name, fno, meal)
M_{QA_3}: QA_Flight(fno, date, to, sold, aid) $\leftarrow$ Flight(fno, date, to),
FlightInfo(fno, sold, cap, aid, type)
M_{QA_4}: QA_Fleet(aid, type, capacity) $\leftarrow$ FlightInfo(fno, sold, capacity, aid, type)

M_{QA_2} and M_{QA_3} are marked as loose mappings and M_{QA_1} and M_{QA_4} as strict ones.

Step 2.

A. We try to define **views** between the schemas of Ontario-Air and Quebec-Air. We observe from $M_{S_A \to OA}$ and $M_{S_A \to QA}$ that the mappings M_{OA_1} and M_{QA_1} are defined over the same set of attributes. Therefore we create the trivial view:

V1: OA_Passenger(pid, name) $\longleftarrow$ QA_Passenger(pid, name)

Also, we observe that the sets of attributes of the mappings M_{OA_2} and M_{QA_2} overlap. Thus we map the one with the more attributes to the one with less attributes using the following view:

V2: QA_Reserve(pid, fno) $\longleftarrow$ OA_Ticket(pid, fno, meal).

Finally, from the mappings M_{OA_3}, M_{QA_3}, and M_{OA_4}, we derive the view:

V3: OA_Flight(fno, date, dest, sold, cap) ⟵
QA_Flight(fno, date, dest, sold,aid), QA_Fleet(aid, type, cap)

where 'to' is renamed as 'dest' and 'capacity' as 'cap'.
B. We now try to define **mapping tables** between the schemas of Ontario-Air and Quebec-Air using views V1–V3. View V1 comes from schema mappings that were strict. Thus, it can stand by itself and we can use it as it is. However, V2 comes from two loose schema mappings. Thus, we understand that we need additional information to use this view: we need a mapping of values in order to fix the 'looseness' of the mappings of attributes 'fno'. Thus, we create a mapping table MT_1(OA.fno, QA.fno). Now we search to find if we can derive more mapping tables. Also, V3 comes from loose schema mappings, both because 'looseness' on the attribute 'fno' and the attribute 'dest'. For 'fno' we have already created a mapping table. For 'dest' and 'to' we create the mapping table MT_2(OA.dest, QA.to).
Step 3. We now ask the user/administrator if she wants to create mapping expressions either on the already created views or any new ones from scratch The user creates a mapping expression from view V1. Let the corresponding mapping expression be

ME: OA_Passenger(pid, name) $\supseteq$ QA_Passenger(pid,name)

As seen in Section 2.1, the mapping expression ME has an executional meaning and does not refer to the structure of the relations.
Step 4. The user populates the mapping tables MT1 and MT2 as shown in Figure 2.
Step 5.
A. Using the mapping expression ME, we create the following rule to ensure consistency over the acquaintance link between Ontario-Air and Quebec-Air databases:

create trigger enforceME
after **insert on** *QA_Passenger*
referencing new as *New* **in** *Quebec-Air_DB*
for each row
begin
insert into *OA_Passenger* **values** *(New.pid, New.name)*
in *Ontario-Air_DB*
end

B. For the maintenance of mapping tables MT_1 and MT_2, we create maintenance rules. Generally the administrator has to decide which kind of rules are needed for the maintenance of a mapping table according to the nature of information that it keeps. Whenever there is a deletion of a tuple in OA_Flight, there is a rule that removes the corresponding tuple in MT_2. The rule flightDeletion given in Section 2.2 captures this interaction with mapping tables.

Step 6. Now suppose the following active behavior that involves two airline peers: DB_A and DB_B agree that whenever a DB_A flight is oversold, a new flight should be created by DB_B to accommodate new passengers. The following is a generic rule that captures this behavior:

```
create trigger AFullFlight
before  update of sold on FlightInfo
        referencing new as New
        referencing old as Old in DB_A
when    New.sold = New.cap
for each row
begin
        insert into Flight values (New.fno, date, New.destination);
        insert into FlightInfo values (New.fno, 0, New.cap, New.aid, New.type)
        in DB_B
end
```

When this rule is customized for Ontario-Air and Quebec-Air peer databases, we get the following rule:

```
create trigger OAAFullFlight
before  update of sold on OA_Flight
        referencing new as New
                    old as Old
        in Ontario-Air_DB
when    New.sold = New.cap
for each row
begin
        insert into QA_Flight values
        (map(New.fno), date, map(New.dest), 0, null)
        in Quebec-Air_DB
end
```

Notice that the action part of the rule above contains an insertion of a new tuple which is a transformation of the updated Ontario-Air flight tuple. This transformation is accomplished using the mapping tables.

Step 7. Quebec-Air is added to the list of acquaintees of Ontario-Air and vice-versa.

Now we give details on the algorithm for establishing an acquaintance in presence of interest groups. Assume that peer N_i issues the following command:

```
set acquaintance to N_j
        belonging to SIG
```

Then the following algorithm is used for completing this request:

If N_j does not belongs to SIG, then follow the simple algorithm of Section 4.1. Otherwise, do the following:

1. For both N_i and N_j, if there are no mappings between the standard schema S_{SIG} of SIG and the actual schemas of S_{N_i} of N_i and S_{N_j} of N_j, then create such mappings.
2. Given the mappings $M_{i \to SIG}$ from S_{N_i} to S_{SIG}, and $M_{SIG \to j}$ from S_{SIG} to S_{N_j}, infer (using algorithms of [7]) a mapping $M_{i \to j}$ from S_{N_i} to S_{N_j}. Use $M_{i \to j}$ to create mapping tables and views between N_i and N_j.
3. Create mapping expressions.
4. Populate the mapping tables, and send a copy of the mapping table instances to N_j.
5. Generate consistency-enforcing rules from mapping expressions obtained in step 2,and generate rules for maintenance of mapping tables.
6. Customize the generic rules of SIG according to the final views and mapping tables obtained in step 2 and activate the customized rules.
7. Add N_j to the list of N_i's acquaintees.

4.3 Abolishing Acquaintances

A peer N_i, may abolish one or more acquaintances with known peers, say $N_{j_1}, \cdots, N_{j_l}$. Again, for simplicity, we assume that acquaintances are explicitly abolished by a user. For this task, the peer language includes the following construct:

`abolish acquaintance to <peer database>`

When an acquaintance is abolished, the various constraints that were attached to it are dropped. Dropping a constraint can be as simple as locally disabling it. However, abolishing an acquaintance can lead to the peer leaving the network if the abolished acquaintance was the only one that the peer had. Therefore dropping a constraint can also be as hard as filling the gap left behind by the vanishing peer. To abolish all the acquaintances of a peer all together, the language contains the construct `leave`.

The following gives details on the algorithm used by a peer to abolish an acquaintance. Assume that peer N_i issues the following command for abolishing an acquaintance:

`abolish acquaintance to` N_j

This is executed as follows:

If N_j is not the only N_i's acquaintee, then do the following:

1. Send a message to N_j to disable any copies of the mapping tables that originated in N_i that it may have.
2. Disable any mapping tables, mapping expressions, and coordination rules that is coupled to the acquaintance $\langle N_i, N_j \rangle$.

Otherwise, do the following:

1. Send a marker to N_j.
2. Steps 2 and 3 as above.

The `leave` command is executed by abolishing all the acquaintances using the algorithm above.

Notice that the "marker" mentioned in the algorithm above is a proxy that the very last peer that was acquainted with a vanishing peer will keep as an indication that the vanishing peer was at some point in the network and has left. We will not elaborate on this marker here. It suffice to mention that this marker can be used to decide on what to do when a query is being answered or an event is being detected while some peers have left the network.

5 Related work

Our architecture adds details to the *local relational model* presented in [3] and extended in [1]. We view peer databases as local relational databases which establish or abolish acquaintances between them to build a P2P network. In [3], each acquaintance is characterized both by a mapping between the peer involved and by a first-order theory that gives the semantic dependencies between the peers. There, it was also indicated that the first-order theory that characterizes the semantic dependencies between the peers can be implemented as ECA rules. In the present paper, we start spelling out the details of this implementation.

That ECA rules constitute an important mechanism for supporting data coordination has been recognized mainly in the context of multidatabase systems, e.g. in [10, 2, 6]. Here, distributed rules involving several databases of the kind "If event E_1 occurs in DB_1, E_2 occurs in DB_2, and E_1 precedes E_2 then carry out transaction T in DB_3" is recognized. Executing such distributed rules requires coordination among peer databases DB_1, DB_2 and DB_3. Therefore, it would naturally be useful to add active functionality to P2P MDBSs. The work in [6] presents a first attempt to study implementation issues for distributed ECA rules as a mechanism of P2P interoperability. However, much work remains to be done on this topic in terms of dealing with the management of seamless addition and removal of peers in the network. Like ours, the proposal for a peer database architecture given in [4], is inspired by the theoretical foundations laid down in [3]. It also mentions the importance of ECA rule for coordinating data exchange between peers. Unlike ours, [4] does not focus on details of how ECA rules can be used for such an endeavour. Finally, in the context of information integration within a common domain, usual solutions (e.g., *global-as-view* and *local-as-view* [9]) that address heterogeneity involve constructing views that reconcile heterogeneous sources in the logical structures. Instead, we consider that metadata is required for data sharing across highly heterogeneous worlds which are often hard to reconcile.

6 Conclusions

We have described techniques for specifying data exchange policies on-the-fly based on constraints (expressed as mapping tables and as generic ECA rules) on the way in which peers exchange and share data. We considered a setting

in which a PDBMS defines and manages its own view of the shared data, and defines its own sharing and coordination policies based on ECA rules provided by interest groups. The on-the-fly specification of data exchange policies is realized by building coordination ECA rules at acquaintance time, as opposed to having them being created at design-time for particular peers by experts. This on-the-fly generation of active rules contributes to our vision of a P2P database technology that is end-user oriented and where establishing acquaintances between peers is done in a flexible, and cost-effective way.

The coordination framework presented here is being extended in various ways. First, we are investigating a complete extension of SQL3 triggers for the peer database setting. For this extension, we are studying a suitable execution model that combines the execution model for SQL3 triggers [8] and the one proposed for the multidatabase context in [6]. Second, the notion of acquaintance type, denoting the level of privileges attached to the acquaintance, is useful in practice. For example, Ontario-Air may have established an acquaintance with Quebec-Air that is of a higher degree of privileges than an existing acquaintance it has with Alberta-Air. We are studying types as a further constraint that characterizes acquaintances. Finally, we started investigating appropriate notions of query processing and transaction management.

References

1. M. Arenas, V. Kantere, T. Kementsietsidis, I. Kiringa, R.J. Miller, and J. Mylopoulos. The hyperion project: from data integration to data coordination. *SIGMOD RECORD*, September 2003.
2. R. Arizio, B. Bomitali, M. Demarie, A. Limongiello, and P.L Mussa. Managing inter-database dependencies with rules + quasi-transactions. In *Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, p. 34–41, Vienna, April 1993.
3. P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *Proc. of the Int'l Workshop on the WEB and Databases*, 2002.
4. F. Giunchiglia and I. Zaihrayeu. Making peer databases interact - a vision for an architecture supporting data coordination. In *Proceedings of the Conference on Information Agents*, Madrid, September 2002.
5. S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In *Proc. of the Int'l Workshop on the WEB and Databases*, 2001.
6. V. Kantere. A rule mechanism for p2p data management. Technical report, University of Toronto, 2003. CSRG-469.
7. A. Kementsietsidis, M. Arenas, and R. J. Miller. Data mapping in peer-to-peer systems: Semantics and algorithmic issues. In *ACM SIGMOD*, 2003.
8. K. Kulkarni, N. Mattos, and R. Cochrane. Active database features in SQL3. In N. Paton, editor, *Active Rules in Database Systems*, p. 197–219. Springer, 1999.
9. M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of the ACM Symp. on Principles of Database Systems*, p. 233–246, 2002.
10. C. Turker and S. Conrad. Towards maintaining integrity in federated databases. In *3rd Basque International Workshop on Information Technology*, Biarritz, France, July 1997.