# Peer coordination through distributed triggers

Verena Kantere*
Ecole Polytechnique Fédérale de Lausanne
verena.kantere@epfl.ch

Maher Manoubi
University of Ottawa
mmano090@uottawa.ca

Iluju Kiringa
University of Ottawa
kiringa@site.uottawa.ca

Timos Sellis
National Technical University of Athens
timos@dblab.ece.ntua.gr

John Mylopoulos
University of Toronto
jm@cs.toronto.edu

## ABSTRACT

This is a demonstration of data coordination in a peer data management system through the employment of distributed triggers. The latter express in a declarative manner individual security and consistency requirements of peers, that cannot be ensured by default in the P2P environment. Peers achieve to handle in a transparent way data changes that come from local and remote actions and events. The distributed triggers are implemented as an extension of the active functionality of a centralized commercial DBMS. The language and execution semantics of distributed triggers are integrated in the kernel of the DBMS such that the latter handles transparently and simultaneously both centralized and distributed triggers. Moreover, the management of distributed triggers is associated with a set of peer acquaintance and termination protocols which are incorporated in the centralized DBMS.

## 1. INTRODUCTION

Emerging distributed applications need to coordinate the exchange of data among autonomous sources. In many cases the managed data are structured and stored in databases. This kind of applications is called peer data management systems (hereafter PDMSs).

Hyperion is an architecture of a PDMS overlay that is described in [4]. Hyperion enables data sharing among peer databases that establish or abolish acquaintances, come on- or off-line while acquainted. Data sharing is performed either by explicit data querying between acquaintees, or by the enforcement of coordination rules that add active functionality on local and remote data. A Hyperion demo that disposes peer query functionality has been presented in [8]. In that demo all the associative functionality for the management of peer acquaintances is realized. However, that demo did not present the automatic data coordination among peers.

In [5] we have presented an approach that employs distributed triggers to enable and coordinate data exchange between peer databases by propagating appropriate updates to acquainted peer DBMSs. To do so, we have extended the syntax of SQL3 to provide a rule language for the definition of Event-Condition-Action (herafter ECA) rules in a PDMS. We delineate a distributed trigger language with simple events, conditions destined to one single peer database, and actions destined to various peer databases. Though simple, this language nevertheless allows realistic coordination of peer data. The execution semantics of the distributed triggers extends the standard semantics of centralized SQL3 triggers by dealing with the autonomous nature of peer databases. The following is a motivating example from the medicine domain that exposes the necessity of distributed triggers in a PDMS.

**Motivating example.** Assume a PDMS in which there are three peer databases as in Figure 1. There is DavisDB - the database of the private doctor Dr. Davis, LabDB - the database of a laboratory and StuartDB - the database of the pharmacist, Mr Stuart. Dr. Davis is collaborating with the lab and the pharmacist, such that the lab performs all the tests and the pharmacist fulfills the medicine prescriptions for his patients. When a regular patient visits Dr. Davis, the latter wants to get automatically from the P2P system all the recent lab tests and medicine prescriptions of this patient. This automatic exchange of data between acquainted peer databases can be realized with triggers that can be distributedly executed on all the involved databases. The following is a distributed trigger:

TRIGGER 1. *When information about a new test is inserted in LabDB about a regular patient of Dr Davis, DavisDB is appropriately updated.*

**Demonstration proposal.** This demonstration presents the real implementation of the distributed triggers that are proposed in [5]. Distributed triggers are implemented literally as an extension of the core centralized triggers of a real commercial DBMS, specifically the well-established, open-source PostgreSQL [1]. This implementation will be incorporated in the Hyperion project and fullfil the requirement for automatic data coordination using generic rules [10].

*This research was conducted while the first author was a Ph.D. candidate at the National Technical University of Athens.


*Proceedings of the VLDB Endowment,* Vol. 3, No. 2

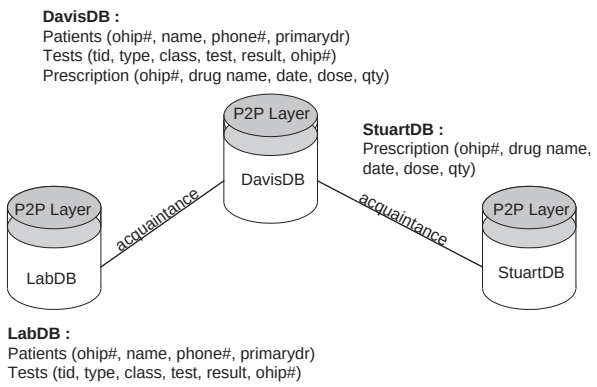


**Figure 1: A network of peer databases**

The demonstration shows the viability of our distributed trigger solution for the automatic coordination of data exchange among autonomous databases. Towards this end, we also employ the instantiation of templates that declare individual a priori requirements for data security and consistency. Such requirements are easily fulfilled by the distributed triggers. Moreover, we will demonstrate the adaptability of the triggers to local or distributed execution semantics according to the overlay state of the peers. Overall, we will present example situations that exhibit the life-cycle of the distributed triggers, as well as their interaction.

## 2. DISTRIBUTED TRIGGERS.

In order to coordinate data exchange in a systematic automatic way, there is a need for a trigger mechanism that can realize a rule of the ECA form that comprises parts from different acquainted peer-databases. This mechanism should realize a distributed trigger language, as well as the appropriate execution semantics.

**Distributed trigger language** To express Trigger 1, we use a language that extends centralized SQL3 triggers [6]. A distributed SQL3 trigger mentions the databases in which event occurrence, condition evaluation, and action execution take place. Distributed triggers are associated at definition time not only with a specific table, but also with a specific peer database. Like in the traditional SQL3 triggers, conditions are SQL queries, and actions are SQL statements to be executed on database instances. Finally, BEFORE and AFTER triggers are defined as in the SQL3 standard.

The event, condition and actions are defined on given peer databases. The trigger action time is BEFORE or AFTER the actual occurrence of the respective SQL statement. The optional DETACHED keyword is added for specialization of the execution semantics of the AFTER triggers and denotes that a trigger should be executed in a distributed AFTER way if involved peers are on-line or in a centralized AFTER way if such a peer is off-line. Also, the condition remains simple, and, thus, it is to be evaluated in a single database. However, the action is a set of separate sets of transactions; each one of these sets is executed atomically in one database. It is obvious that in case all parts and subparts of the trigger are declared to be executed in a single database, our extended semantics are reduced to the centralized SQL3 semantics.

Trigger 1 is defined as follows:

```
CREATE TRIGGER testInsertion
AFTER INSERT ON Tests
    REFERENCING NEW AS NewTest IN LabDB
FOR EACH ROW
WHEN EXISTS SELECT P.ohip#
                FROM Patients P
                WHERE P.ohip# = NewTest.ohip#
                      AND P.primarydr ='F'
                IN DavisDB
BEGIN
        INSERT INTO Tests VALUES NewTest IN DavisDB
END
```

**Distributed trigger execution semantics.** We have extended the processing mechanism of the centralized SQL3 triggers (without considering constraints) in a way appropriate to deal with the heterogeneous and autonomous nature of peer databases. We deal with autonomy by considering the transient character of peer databases.

The trigger processing mechanism consists of an execution semantics, a set of termination protocols, and a set of protocols for each one of the following tasks: establishing and abolishing acquaintances, connecting to and disconnecting from a P2P network, and joining and leaving a P2P network. For simplicity, we call the third component acquaintance protocols. The execution semantics specifies how the distributed triggers are executed. For details on the execution of all kinds of triggers and especially the DETACHED triggers see [5]. Termination protocols are similar to those used for commitment in distributed DBMSs [7]: Whenever a peer involved in the processing of a trigger disconnects, the rest of the peers should be able to gracefully terminate trigger processing. Finally, acquaintance protocols address how the trigger execution process at the disconnected peer recovers when the peer reconnects.

## 3. PEER DATA COORDINATION

A big concern in a database environment is security and consistency. On the opposite of distributed DBMSs, P2P DBMSs do not require or ensure by default either restricted access or consistency of data in an inherent way. Concerning security, the acquaintance of two peers w.r.t. to a part of their schemas implies by default full access to the acquainted schema part by the acquaintee. Concerning consistency, peer autonomy retracts the guarantee (of distributed databases) that data in acquainted peers are consistent at any time.

Distributed triggers can be employed in order to customize the security and consistency degrees according to the requirements of the PDMS or the individual needs of peer databases. For example, it is the case that a doctor would not like his data on prescriptions to be changed by another acquainted database, even if the latter has access to the respective part of the schema of the doctor's database (as part of their acquaintance). Concerning the network of Figure 1, the table Prescriptions of DavisDB should not be changed by its acquaintees, without first notifying Dr Davis; the latter will or not approve this change. As another example, a laboratory would not like updates on the local test data performed by doctors without first checking the oncoming update for mistakes or inconsistencies. In Figure 1 LabDB would like to be notified and locally approve updates or inserts on the Test relation, to be done by acquainted databases.

```
<template definition> ::=
CREATE COORDINATION TEMPLATE <template name>
FOR <db name>
BEGIN <coordination declaration>+ END
<coordination declaration> ::=
<table name>:<single declaration>+
<single declaration> ::=
[NO] <distributed trigger>(<trigger part>+ [| <time>])
<distributed trigger> ::= BEFORE | AFTER |
DETACHED AFTER
<trigger part> ::= EVENT: | ACTION: <db operation>
<db operation> ::= INSERT | DELETE | UPDATE
```

**Figure 2: Structure of the basic template for P2P database coordination**

In the same way, acquainted databases may desire different degrees of consistency for their peer data. Some data between acquaintees should be always consistent, whereas other may tolerate inconsistency for some limited or not time. Since peer autonomy precludes the solution of consistency enforcement on peer data (e.g. a peer database may be offline), specifications on alternative actions of database operations on local/peer data or notifications should be enabled. For example, Dr Davis would probably like to be notified if a new prescription inserted in DavisDB is not propagated to StuartDB in a limited time, so that he can take an alternative action, such as contacting another pharmacist or handing the prescription to the patient. Moreover, probably Dr Davis would like different or additional actions in case that consistency can be enforced right away or not.

**Coordination templates.** Coordination of data that require different degrees of consistency and security can be achieved using the distributed triggers. The latter can be employed in order for peers to denote in a declarative straightforward manner their preference degrees of security and consistency enforcement for special parts of their peer schema and data. Thus, parts of the peer schema that is involved in an acquaintance can be associated a priori, at the time of the acquaintance or before, with a set of security and/or consistency requirements.

It is easy to create templates that can be used for this purpose. The structure of such a basic template is presented in Figure 2 and with the help of it we can declare what types of distributed triggers should or should not be imposed in case of an acquaintance. Of course it is possible to extend the structure of the basic template in order to support declarations customized for each acquaintance of a peer or to support events other than database operations. We call the declarations that can be created with such a template *coordination declarations*. Note that these declarations refer to database operations that are candidate events or actions for distributed triggers. The following are example templates for the peer databases:

```
CREATE COORDINATION TEMPLATE Template1
FOR LabDB
BEGIN
Tests:          BEFORE (ACTION: Insert, Update, Delete)
END
CREATE COORDINATION TEMPLATE Template2
FOR DavisDB
BEGIN
Prescriptions: BEFORE (ACTION: Insert, Update)
               DETACHED AFTER (EVENT: Insert | 1 Day)
END
```

```
CREATE COORDINATION TEMPLATE Template3
FOR StuartDB
BEGIN
Prescriptions: NO DETACHED AFTER (ACTION: Delete)
END
```

The above templates define the a priori security and consistency requirements of the peer databases of Figure 1. Database LabDB declares that no external (i.e. by acquaintees) operations can be performed on its data about tests without any prior notification of the local administration. Similarly, DavisDB does not want any alterations in his prescriptions without his approval. Also propagations of his prescriptions to the pharmacist must be performed within a day, otherwise he may want to take a different action. Finally, in order for Mr Stuart to avoid to give out obsolete or withdrawn prescriptions, StuartDB declares that there should be not any external deletions of prescriptions made offline (i.e. by detached after triggers).

**Flexible acquaintances based on templates.** Templates can be employed in order for a peer to adopt alternative acquaintance attitudes concerning consistency but also security, according to the overlay status of itself or its acquaintees. Mr Stuart may wish to use Template3 defined above for periods that he is working and the following when he is on vacation:

```
CREATE COORDINATION TEMPLATE Template4
FOR StuartDB
BEGIN
Prescriptions: NO DETACHED AFTER (ACTION: Insert)
END
```

In this case, Dr Davis may wish to be notified before he inserts a prescription, in order to hand the prescription to the patient, instead of automatically propagate it to the pharmacist:

```
CREATE COORDINATION TEMPLATE Template5
FOR DavisDB
BEGIN
Prescriptions: BEFORE (EVENT: Insert, Update)
END
```

Note, that local and distributed triggers are managed in a transparent way by the extended peer DBMS.

# 4. IMPLEMENTATION

We have chosen to implement the proposed distributed triggers as an extension of the trigger mechanism of an existing active DBMS. Specifically, we have chosen PostgreSQL 8.3.6[1, 9], since it is open-source and widely used. Overall, our extension in the kernel of the aDBMS includes:

- The definition of triggers that are triggered by an event (limited to a database operation: insert/delete/update) on the local database, but the corresponding action is to be executed on a remote acquainted database.
- The execution of actions on the local database that are defined by a remote trigger (i.e. a trigger that is defined by an acquainted database).
- The definition of the execution mode of the trigger with respect to the databases that are involved.

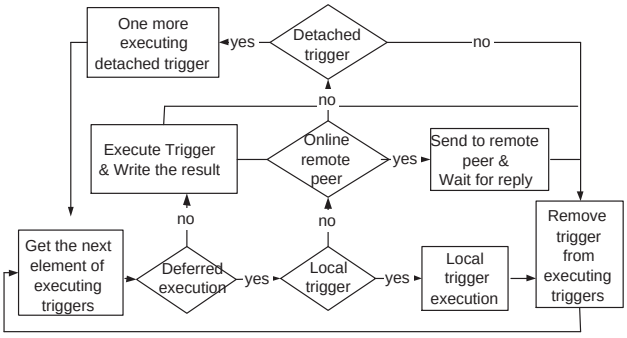


**Figure 3: Execution flow of detached triggers**

- The addition of a new execution mode for the pair event-action: beyond adapting the *deferred* and *immediate* modes, we implement the new mode: *detached*.
- The execution of the distributed trigger according to the corresponding execution mode. This involves the adaptable execution of a trigger in an immediate or deferred mode, automatically, if the detached mode is defined.

The above extension of PostgreSQL is implemented as a modification in the $(a)$ module that allows the definition of a trigger, but also in the modules that parse a trigger definition in order to proceed with execution (for example storage and activation); $(b)$ the execution mechanism, such that: $(i)$ execution of remote actions is enabled, $(ii)$ the new execution mode *detached* is supported; and $(c)$ the catalog of the centralized DBMS is extended, such that it supports storage of data and meta-data that refer to remote databases.

**Extending the trigger execution mechanism.** We extended the trigger execution mechanism such that:

- Triggers can be executed in detached mode.
- Functions/actions on remote databases can be executed.

Beyond extending the two established trigger execution modes: BEFORE and AFTER, we have implemented the new DETACHED mode. In the last case, if an involved database is off-line, trigger actions are executed in a new transaction when this database comes on-line. Databases that either manage a detached trigger or have to execute related actions, keep respective logs of the tasks to be fulfilled when the appropriate combination of databases is on-line again. Figure 3 summarizes the general execution flow of triggers in detached mode. In the current implementation the default mode of distributed triggers is the detached mode.

**Implementing communication** In order to make distributed triggers operational for peer databases located in computers with physically separate networks, we add a virtual DBMS layer on each peer. Beyond offering a GUI for easy declaration of the distributed triggers using the templates, it deploys the appropriate triggers, tables, views and functions in the respective peer database. The GUI also enables the initiation of local database operations. The virtual DBMS layer maintains the P2P communication among databases during object deployment and runtime transactions and termination protocol.

The communication uses configuration functions implemented employing PL/PROXY [2] and PL/PGSQL [3]. The functions hold information about acquainted peers such as the database name, the host and the password. On trigger execution, the remote database name is matched with the cluster name, and the connection information is returned and used to connect to the remote database.After the connection establishment, the trigger passes all the necessary data as argument to a function in the remote database.

## 5. DEMONSTRATION

The aim of the proposed demo is to make apparent the viability of our solution for data coordination among acquainted peer databases employing the distributed triggers. Using a PDMS scenario from the medicine domain, as in our motivating example, we will present the interaction of several real peer DBMSs through the employment of distributed triggers.

We intend to show how peers can define or annul automatic data exchange through the procedure of acquaintance establishment or abolishment, respectively. Beyond this, we will present a basic and some more elaborated templates that can be easily instantiated even by end-users in order to express security and consistency requirements for different peer schema parts. Moreover, we intend to show how distributed and centralized triggers can be executed simultaneously and interact in our extended PostgreSQL DBMS. Finally, the demonstration will include examples of leave of peers that require termination of distributed triggers as well as rollback of their actions already executed. These situations will display the effectiveness of our termination protocols.

## 6. REFERENCES

[1] PostgreSQL. http://www.postgresql.org/.
[2] PL/PROXY. http://plproxy.projects.postgresql.org/.
[3] PL/PGSQL. http://www.postgresql.org/docs/8.3/static/plpgsql.html.
[4] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos. The hyperion project: from data integration to data coordination. *SIGMOD Record*, 32(3):53–58, 2003.
[5] V. Kantere, I. Kiringa, Q. Zhou, J. Mylopoulos, and G. McArthur. Distributed triggers for peer data management. In *CoopIS*, 2006.
[6] K. Kulkarni, N. Mattos, and R. Cochrane. Active database features in sql-3. In N. Paton, editor, *Active Rules in Database Systems*, pages 197–219. Springer Verlag, 1999.
[7] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, Upper Saddle River, 2 edition, 1999.
[8] P. Rodriguez-Gianolli, M. Garzetti, L. Jiang, A. Kementsietsidis, I. Kiringa, M. Masud, R. Miller, and J. Mylopoulos. Data sharing in the hyperion peer database system. In *VLDB*, 2005.
[9] M. Stonebraker, E. Hanson, and C. Hong. The design of the postgres rules system. In *dataeng*, Los Angeles, CA, feb 1987.
[10] D. Zhao, J. Mylopoulos, I. Kiringa, and V. Kantere. An eca rule rewriting mechanism for peer data management systems. In *EDBT*, 2006.