Realisation of Active Multidatabases by Extending Standard Database Interfaces

Christopher Popfinger

Institute of Computer Science - Database Systems
Heinrich-Heine-University Düsseldorf
D-40225 Düsseldorf, Germany
popfinger@cs.uni-duesseldorf.de

Abstract

The continuous progress in communication, networking and database systems demands a new perception of data processing. A growing amount of network-based applications requires access to a multitude of mostly autonomous data sources located in a heterogenous software and hardware environment. Advances in database design offer new possibilities for the integration of autonomous local database systems. Active component systems do not only provide access to stored data, but significantly control business rules and integrity constraints by working cooperatively with each other while retaining the greatest possible extent of local autonomy. We introduce ideas and concepts for the realisation of *active multidatabases* by extending standard database interfaces.

1 Introduction

Nowadays, Internet and telecommunication provide easy access to stored data worldwide and web applications enable consumers and users to easily manage their data from almost any terminal connected to the web. However, in practice, the required data is rarely stored in a single well designed database, but rather spread among a various number of heterogeneous data sources, which emerged mostly independently from each other with a high grade of local autonomy to fit special needs of local applications.

Consider, for example, a scenario which can easily be found in the real world. A directory service controls access to resources (e.g. login for a web interface or rights management for various data sources), while the information to be processed is stored in additional heterogeneous data sources (e.g. relational databases, XML documents or textfiles). To access these data sources the concepts of multidatabase systems propose a global component, the Global Transaction Manager (GTM), which integrates the sources and coordinates global transactions to the local component systems.

One of the major problems herein is, as part of the transaction management, the secure simultaneous modification of data in several component systems, which requires an atomic commitment protocol. As shown in [2] this is impossible without constricting the local autonomy of the data sources. The GTM is the sole component, which is responsible for the proper and secure execution of global transactions, while the component systems simply act as passive sources for data.

By migrating the concepts of active database systems to multidatabases, we develop active component systems, which do not work seperated but directly interact with each other, assisting the GTM in transaction management. Global business rules and integrity constraints are exported to the local databases, which are now partially responsible for the correctness of global transactions. To retain the greatest possible extent of local autonomy, we propose the usage of standard database interfaces like trigger or stored procedures, extended by further functionalities.

2 Active Component Systems in Multidatabases

In practice, there are many considerable examples for the usage of multidatabase systems. Especially web applications often need access to data stored in multiple local data sources, to be displayed, modified and written back. The GTM is the vital modul for proper and secure integration of the autonomous local database systems (LDBS). Applications can transmit global queries to the GTM, which distributes them to the appropiate LDBSs, organizes the results and returns those to the applications. Within this classical notion of multidatabase systems, the component systems do solely have passive functionality. They provide access to the data via standardized query languages (e.g. SQL), whereas communication thereby is established via database servers, which likewise provide connections from applications to the database system via standardised interfaces. Normally the isolated component systems do not have any knowledge of other LDBS of the multidatabase.

The GTM takes full responsibility for the correctness of transactions and their related data. Local autonomy of the component systems (especially communication and execution autonomy) extremely affects global transaction processing. For this reason particularly simultaneous updates in multiple LDBS can result in inconsistencies, if, for instance, conflicts with local transactions force a rollback in a component or a system failure impedes the transaction processing. An atomic commit for global transactions in multidatabases can only be realised by violating the local autonomy (e.g. a visible prepare to commit state), limiting the type of transactions allowed or using different transaction models [2]. In the following we like to introduce a concept to increase the safety of global transactions without decisive limitations to the autonomy of local component systems. Primarily we take a look at database systems with a database management system (DBMS), to which the concepts of active databases can be applied.

Active database systems assist applications by migrating reactive behaviour from the application to the DBMS. Therefore they are able to observe special requirements of the application and to react in a convenient way if neccesary [4], to preserve data consistency and -integrity, generally over multiple tables inside a single database. Thus the database system is able, with restrictions, to check data and monitor business rules. It is no longer a passive supplier of data but participating at processes actively. The DBS is jointly responsible for the proper execution of global transactions and the preservation of a consistent global data set. Lightweight applications can be developed by outsourcing a portion of the rules to the data source. The inclusion of active behaviour in relational database systems is not particularly new and most commercial systems use triggering mechanism, whereas the execution of rules is mainly activated by operations on the structure of the database (e.g. insert or update a tuple), than by user-defined operations [4].

Multidatabases often consist of DBMS driven component systems, which generally provide active functionality to preserve local quality of data, for instance by checking referential integrity or the usage of triggers, yet limited to the isolated local databases themselves. The execution of a global transaction mostly requires and modifies data from several LDBSs, which necessitates integrity and consistency checks. Up to now, in multidatabases this is done by the GTM, thus a system component respectively an application. Now it is possible for component systems to participate actively in transaction management and thereby ease the workload of the GTM. They have knowledge of other components of the multidatabase and are abled to check global integrity constraints, while retaining the greatest possible extent of local autonomy. This can be achieved by using extended standard interfaces of the database, which will be discussed in the next section.

3 Realisation via Extended Standard Database Interfaces

The realisation of active component systems with least possible restriction to local autonomy could be achieved via extended standard interfaces of databases. Therefore we basically take a look at *trigger* and *stored procedures*. Especially in relational database systems, these functionalities are already used for a long time, but until now there is the problem of establishing communication between the heterogeneous and autonomous component systems, to let them coordinate their actions, similar to a homogeneous distributed database system. By now communication mainly occurs between the GTM and the local sources, while direct arrangements between heterogeneous component systems could not be realised due to technical reasons.

Further progresses in database research and the provision of platform independent software solutions offer new possibilities. Manufacturers of commercial database systems expand their systems with new functionalities, like for example the direct usage of Java or the execution of Java-Applications in trigger and stored procedures [3]. In the following we like to concentrate on Java extensions, though other extensions are conceivable, which basically provide the same functionality.

By the use of these extensions to existing interfaces, the communication problem between component systems is basically solvable. We are able to establish connection to other LDBSs via standardised database interfaces (e.g. JDBC), as well as to send queries or to update the remote data base directly from triggers and stored procedures. Thus, the component systems are able to communicate with each other and so exchange status information or even data items, enabling them to basically coordinate global transactions.

Let us take a look at the following examples with two active component systems A and B, being integrated in a multidatabase via a GTM:

- **Referential Integrity:** A and B store information about persons. Database A manages access rights of users for various resources, while B contains personal information (e.g. address, banking account, etc.). Insertions in B concerning a user shall only be possible, if all security relevant entries for that person exist in A. When inserting a new tuple into B, a trigger is activated (`BEFORE INSERT`), opening a connection to A by calling a Java extension to check the existence of the required user data. In this case the new tuple can be inserted, otherwise the trigger mechanism of the DBMS interrupts the insertion.

- **Simultaneous Update:** A global transaction inserts or updates data simultaneously in A and B or a modification in A implies a modification in B. When inserting/updating a tuple in A, a trigger is activated (`BEFORE INSERT/UPDATE`), which establishes a connection to the database server of B by calling a Java extension and inserts or updates remote data stored there. Only if this operation succeeds, the modifications are stored permanently in A.

Regarding a common approach to a knowledge model for active rules with the components *event*, *condition* and *action* (ECA), we can classify the call of a Java extension from a trigger on the one hand as a part of the *condition* component which regards the context of an event [4]. We can specify the evaluation of a condition (among others) as the result of a query, an insert or update, or a deletion of tuples in another LDBS, which determines the type of the sequenced action. On the other hand, a call of a Java extension can be executed as a part of an *action*, triggered by a previous evaluation of a condition. A precondition for that is the applicative support of the Java interface in the rule language for both components (*condition* and *action*) of the knowledge model.

As it is basically possible to call stored procedures and thus Java code directly from SQL statements, the operational interface of such an active LDBS provides additional status information and indirect transaction operations (i.e. transactions between two component systems of a MDBS) for use with both an extended base transaction model and a service interface model as mentioned in [1].

Due to the flexible use of the interface extension, we can give a multitude of examples of use. A component system can query directly the states of other LDBSs and exert influence on them, without accepting decisive limitations to its local autonomy. Thus, we are able to implement and check integrity constraints and business rules of the GTM directly in the LDBS, whereas the communication is realised via existing interfaces, which are also used by the GTM to communicate with the MDBS components. To the LDBS the query of another component system looks similar to a common query of an application or a GTM. We want to take a closer look at the profits of this concept in the following section.

4 Profits

The proposed concept offers a good start for future research and we still have to determine how exactly the extensions are useful for transaction management in multidatabases. But up to now we can work out some basic advantages:

- **Checking the state of a remote LDBS:** An active component system can directly query another data source of the multidatabase. This status information can potentially be used in scheduling global transactions, preserving global atomicity or avoiding global deadlocks.
- **Manipulating a remote data base:** By establishing direct connections between LDBSs and executing any of the allowed transaction operations, an active LDBS can modify data of another component system of the MDBS directly. This may be useful when monitoring global business rules or checking global constraints (i.e. referential integrity of global data).
- **Few limitations to local autonomy:** Many DBMSs already provide active behaviour in the form of trigger and stored procedures as a main feature. Generally, no changes to the software are neccesary, and so no violation to the design autonomy of the local data source occurs. Furthermore, a submitted subtransaction is locally scheduled by the LDBMS. Since the Java calls are embedded straight into the DBMS, we are able to delay or abort transactions, depending on the state of another data source or the result of an indirect transaction without 'unnaturally' limiting execution autonomy. This could be used for global serializability or deadlock detection/prevention.

Being able to actively communicate, the local DBMSs are aware of each other and are basically enabled to coordinate their actions. Thus, traditional techniques for ensuring transaction atomicity and consistency in homogeneous DBS can possibly be applicable to multidatabase systems.

5 Problems

As shown above, it is unsure by now, if the proposed concept can be useful to solve or facilitate the main problems of multidatabase transaction management, which are *Global Serializability*, *Global Atomicity and Recovery* and *Global Deadlocks* [1]. Possibly, it cannot be guaranteed, for example, that a simultaneous update operation is completely and consistently executed (cp. atomic commitment protocols). While status information operations are more or less uncritical, the execution of a transaction modifying data can result in inconsistencies. If the execution of a trigger, which in turn executes an indirect transaction to modify data, gets interrupted after data has already been stored in other LDBSs, then these modifications cannot be rolled back. The trigger mechanism of the DBMS is not able to run any recovery algorithms. Furthermore, there is a couple of problems exclusively related with active component systems, in particular:

- **Identifying 'Master'-LDBS:** If we want LDBS to execute indirect transactions, we have to identify 'master' component systems, which receive global transactions from the GTM and perform modifications to the remaining LDBSs, particularly if there is more than one active database, which is basically able to.
- **Cycles:** While evaluating triggers, we can think of an execution order, where updating a DBS causes modifications (i.e. write operations) in other DBSs and creates, at the worst, a cycle of trigger transactions.
- **Rule Migration:** When migrating global business rules and constraints to the LDBS, we have to decide where to implement which set of rules. Rule migration could be facilitated by appropriate extensions to modelling languages for multidatabases.

6 Conclusion

The realisation of active multidatabases by using extended standard interfaces offers a multitude of new possibilities for securing data quality and integrity. Active component systems work cooperatively by communicating over standard database interfaces without greater limitation to their local autonomy. Thereby it is possible to query and modify states of other component systems from within triggers, principially both while evaluating a condition and in trigger actions. Stored procedures can be used at any time to receive system information of a remote data source or to activate update operations and consistency checks over multiple local data sources. The provision of an extended set of DBMS-embedded status information and transaction operations potentially offers alternatives to existing multidatabase transaction models. Future research has to be done to determine the practical use of the concept for serializability, atomicity and deadlock detection and how the related problems can be solved.

References

[1] Breitbart, Y.; Garcia-Molina, H.; Silberschatz, A.: Overview of Multidatabase Transaction Management. In VLDB Journal 1(2), pages 181-293, 1992.

[2] Mullen, J.G.; Elmagarmid, A.K.; Kim, W.: On the Impossibility of Atomic Commitment in Multidatabase Systems. In Ng, P., Ramamoorthy, C., Seifert,L. and Yeh, R., editors, *Proc. of The International Conference on System Integration (ICSI'92)*, pages 625-634. IEEE Computer Society Press.

[3] Oracle Technology Network: Java in Oracle9i Database. http://otn.oracle.com/tech/java/java_db/content.html, March 2003.

[4] Paton, N.W.; Diaz, O.: Active Database Systems. In ACM Computing Surveys 31(1), pages 63-103, 1999.