INTRODUCTION

Database management systems (DBMS) are becoming part of environments composed of large-scale distributed heterogeneous and networks of autonomous, loosely coupled components. In particular, federated database management systems (FDBMS) can be seen as networks that integrate a number of pre-existing autonomous DBMS which can be homogeneous or heterogeneous. They can use different underlying data models, data definition and manipulation facilities, transaction management, and concurrency control mechanisms. DBMS in the federation can be integrated by a mediator providing a unified view of data: a global schema, a global query language, a global catalogue, and a global transaction manager. The underlying transaction model considers, in general, a set of transactions synchronized by a global transaction. Synchronization is achieved using protocols such as the Two-Phase Commit protocol. FDBMS applications are built upon this global environment, and they interact with the mediator to execute global database operations (i.e., operations that can concern various DBMS in the federation).

In order to illustrate intuitively the use of FDBMS, consider a financial context where a shareholder has two bank accounts in Mexico and in France managed by database applications. This person needs to execute banking operations either accessing the accounts independently or making applications cooperate to have a global view of his/her financial situation. Clearly, an FDBMS application would fulfill these requirements, letting to execute transparently banking global (e.g., consult of the global credit) and local operations (e.g., withdraw 1000 pesos from the account in Mexico). Assume now that shareholders need specific operations to be executed timely when certain conditions come up:

- WHEN the dollar price changes in France,
  IF my account in France has more than 100000 euros
  THEN send an e-mail to advise me to buy dollars.

- WHEN money is withdrawn from my bank accounts,
  IF my global credit is less than 1000 pesos
  THEN abort the operation.

Even if some DBMS provide some active capabilities (triggers), federated database systems are often limited when considering the encoding of the behavior of data and the reaction of the system(s) to data changes. Operations are executed after explicit request submitted by a user or an application. These systems may be referred as passive FDBMS, in opposition to active FDBMS that are able to execute automatically predefined actions in reaction to specific events when some conditions are satisfied.

As in active DBMS (Paton, 1998), the major idea in active FDBMS is to add a reactive mechanism as ECA rules. Rules are processed in a specific environment defined as an active FDBMS application including global transactions. In such environments, possible uses of active rules are numerous, for instance, are view maintenance, global schema updates, verification and validation of global integrity constraints, notification, application, component integration and cooperation, and so forth.

The use of rules in FDBMS applications implies at least three challenges. First, the inherent heterogeneity of the system imposes the need of a flexible rule execution model adaptable to the characteristics of the participating DBMS. Second, the active mechanism must deal with the autonomy of both the global system itself and the participating DBMS. In an FDBMS, DBMS can keep their communication and execution autonomy. Thus, they may share or not control information, and they may continue to control their execution at any moment, independently of the federation. They can commit or abort local transactions at any time, and this can affect the execution of global operations. Third is events management stemming from different contexts. Communication protocols are needed to observe events from their sources (DBMS) and signal them to consumers (rules). Events are messages containing information about the federation and its components. Therefore, they should be processed respecting information consistency, legacy, and performance needs.

For providing active capabilities within FDBMS, it is necessary to go beyond what has been proposed and developed in the context of the active databases domain (Chakravarthy, Le & Dasari, 1998). This article proposes an event service and a rule service that cooperate to execute active rules within an FDBMS.
THE SERVICES APPROACH

Active FDBMS require a federation-wide mechanism for event handling and reaction execution. In such a context, it must be possible to detect events to make them visible to other components of the federation. Events can be either observed or raised by each participating DBMS. It must also be possible to couple the execution of actions with the execution of FDBMS applications.

From the wide variety of proposals, distribution has not been widely introduced in active database environments. Furthermore, the experience of research works shows that active capabilities are potentially useful and pertinent to answer to non-database applications. However, the architecture of these systems does not allow providing active functionalities able to fulfill the requirements of every application. Therefore, we provide active capabilities as services instead of a monolithic system. Services are independent components, and they are the core of an extensible mechanism. The problem is how to unbundle the mechanism so that a full-functioning and lean active mechanism remains. Figure 1 shows processes that have to cooperate for getting an active FDBMS, each of them based on a model.

The knowledge model describes how to define, represent, and administrate data and rules. The FDBMS data model defines the data structure and the operations that can be executed over data. The rule data model defines the Event, Condition, and Action parts of the rule. Conditions and actions are specified considering the FDBMS data model. Hence, they may implicitly concern several DBMS in the federation.

Event definition is supported by an event model. It specifies the set of event types representing the significant situations that have to be observed across and within the FDBMS (event type model) and the way such events have to be detected, composed, and notified (event management model).

Figure 1. Active FDBMS

An FDBMS application is executed according to the execution model of the system (e.g., global transactions). Similarly, rules are executed according to an execution model that characterizes the coupling of event consumption, condition evaluation, and action execution within and across FDBMS or global transactions.

Implementing the above models must not demand the complete re-definition of the federation and applications. Therefore, we first isolated the active mechanism from the FDBMS control. Then, we isolated the event mechanism from the rule execution one to specify two independent services.

The event service detects complex situations within clients (DBMS, applications) and notifies them automatically to other clients. The rule service executes asynchronous reactions to events. The technical challenge of this approach is the amalgamation of communication, database, and active aspects (Bützingsloewen et al., 1998).

The event service administrates event managers that detect, produce, and notify events. The FDBMS and application execution are monitored to detect and produce (i.e., recognize, order, and compose) events described by rules. Then, events are delivered to the rule service under different modes and communication protocols depending on rule execution models. The rule service is responsible for rule execution that consists of two phases:

- The planning phase that builds an execution plan according to different strategies (Coupaye & Collet, 1998).
- The execution of one rule: the evaluation of the Condition part and the execution of the Action, if the Condition is satisfied.

A parametric interface to specify interaction among the services of the FDBMS has been proposed in Collet, Vargas-Solar, and Grazziotin-Ribeiro (2000) and Grazziotin-Ribeiro (1999).

EVENT SERVICE

The event service defines a framework for specifying and generating (implementing) event managers. It supports two meta-models for specifying event managers (i.e., defining their event type system and their event management model).

Event Type Description

The Event Type Meta-Model (Collet et al., 1998; Vargas-Solar, 2000; Vargas-Solar & Collet, 2002) provides con-