ABSTRACT

This paper describes an object-oriented approach to managing federated database systems. In particular, we describe an object-oriented execution model for a federated database system and illustrate the feasibility of this model by describing how it manages a number of federation operations.

1. INTRODUCTION

A federated database system (FDS) has been defined to be a collection of cooperating database systems which are possibly autonomous and heterogeneous [Sheh90]. The intent is for an individual database system to retain its autonomy, whilst at the same time participate in a federation if it wants to. This participation should not affect the local users of the system.

The database systems participating in the federation could be homogeneous, in which case they are all designed and operate identically, or they could be heterogeneous with respect to several aspects such as data models, designs, semantics, constraints, among others. While heterogeneity brings about complexities not present in a homogeneous environment, a consequence of the local autonomy requirement is that it enables a database system to join or leave a federation without any restrictions. This makes the task of developing a FDS even more difficult.

Recently several research and development activities on FDSs have been reported (see, for example, [Elmargamid90]). The work by Omololu et. al. [Omololu90] has developed software which addresses some of the problems that must be overcome in developing a FDS. In addition, the components identified in the system by Fiddian et. al. [Fiddian92] can be adapted to serve as the basis for some of the objects defined in this paper. Although some promising results have been obtained, several tasks such as multiuser updates, are yet to be carried out successfully. It has been more or less agreed that the solutions to interconnecting multiple database systems flexibly and with full interoperability are a generation away [Silberschatz90].

There are various types of heterogeneity that need to be addressed for a FDS. The major issues involved in interconnecting heterogeneous components as identified in [Sheh90] are the following:

(i) Schema (or data model) Heterogeneity: Not all of the databases in a heterogeneous architecture are represented by the same data model. One method of tackling this heterogeneity is to create an integrated conceptual model which enables the users to work efficiently with the federated system's data in a uniform manner. In order to do this, translators which transform the constructs of one data model into those of another are being developed. This integrating software must also take account of the semantics of the data, if this is available [Quaisha92]. If the federated users want to establish their own views as to what data descriptions are included then schema translators will be needed between the different schema representations supported locally.

(ii) Transaction Processing Heterogeneity: Different DBMSs may utilize different algorithms for transaction processing. Work is being directed toward integrating the various transaction processing mechanisms. For example, techniques which integrate locking, timestamping, and validation mechanisms are being developed.

(iii) Query Processing Heterogeneity: Different DBMSs utilize different query processing and optimization strategies. One of the research areas here is to develop a global cost model for distributed query optimization. This also needs to be linked to the schema integration software as they can create valuable information in the integrated schema for the optimiser [Quaisha93].

(iv) Query Language Heterogeneity: Different DBMSs will utilize different query languages. Even if the DBMSs are based on the relational model, one could use SQL and the other Relational Calculus. Standardization efforts are under way to develop a uniform interface language.

(v) Constraint Heterogeneity: Different DBMSs enforce different integrity constraints which are often inconsistent. For example, one DBMS could enforce a constraint that all employees must work at least 40 hours while another DBMS may not enforce such a constraint. These differences need to be reconciled.

(vi) Semantic Heterogeneity: Data may be interpreted differently at different components. For example, the entity address could mean just the country within one component while another component could interpret it to be the number, street name, city name and country. It has been recognized that semantic heterogeneity is very difficult to handle [Corcone90]. Standardization efforts are needed to resolve such inconsistencies.

Much of the previous work has focussed on developing solutions to the separate problems in handling heterogeneity such as transaction processing, schema integration and translation, and semantic heterogeneity. Furthermore, many efforts have proposed the object-oriented data model to be the global data model.

Because of the approach normally taken, it will be difficult to handle more than one or two types of heterogeneity. At the extreme case, the different DBMSs may utilize different data models, use different transaction processing algorithms, and have syntactic and semantic differences. To overcome this problem, some efforts have concentrated on using the meta-techniques, for example Fiddian92, Quaisha93]. This approach which has produced fruitful results still needs a global architecture to manage the federation efficiently. In this paper we propose that in order to provide a seamless integration of the different autonomous DBMSs, one must develop an execution model of the heterogeneous environment.

One solution which appears to show much promise is the distributed object management systems (DOMS) approach [Nicolas93]. With this approach, each DBMS, the database, schemas, and the data in the database are treated as objects. Objects communicate with each other by exchanging messages. That is, the various components are encapsulated as objects in the distributed object management system.
The object model of the DOMS could be used to serve as the global data model for the federation.

This paper describes a DOMS to model the federated environment. First, it shows how the entities in such environment are classified and handled. Then, it describes how the object model of the DOM can be used to handle the operations at the federation level.

The organization of the paper is as follows. In Section 2 we present the application of object-oriented approach to model a distributed database environment. This is followed in Section 3 by examples on how some of the usual database operations are handled in a DOMS. We conclude by giving some advantages in using this approach.

2. DISTRIBUTED OBJECT MANAGEMENT SYSTEM FOR THE FEDERATED ENVIRONMENT

The execution model of the heterogeneous distributed environment, is essentially a distributed object management system (DOMS). Note that such systems are becoming very popular for the interoperability between tools and resources. Furthermore, standards such as CORBA (common object request broker architecture) are being developed for distributed object management [Nicol93]. With this approach, a DBMS as well as database is encapsulated as an object. Below we describe the essential points of the object-oriented model of the DOMS that we have considered.

In our model every entity is an object. That is, an object could be a federation, a node, a database, schema, or a DBMS. We group collections of objects with similar properties into classes. The classes form class hierarchies; we support inheritance and encapsulation. The properties of a class are specified by instance variables. Instance variables in turn are themselves objects which may consist of other objects or even primitive ones such as strings or integers. Objects also include methods which consist of codes used to manipulate the object or return some state from it. Invocation of a method from outside is by means of messages. Thus, a collection of messages to which an object responds, forms the interface of the object.

By using this approach, the DOMS maintains autonomy, independence, and provides a natural and clear interface between object entities in the federation. This allows the federated environment to accommodate a wide variety of vendor products, DBMSs, schemas etc. which are abundant in today's heterogeneous environment. There is also a future possibility of gaining interoperability with non-database information systems.

We will illustrate the management of this environment with an example. Figure 1 represents the environment partially. The classes include FEDERATION and NODE. The instances of the FEDERATION class are the various federations. Each federation has the following instance variables: the federation-ID, the federation-name, the collection of nodes which form the federation, the federated schema, the federated administrative policy, and an administrator or group of administrators if there is one.

The NODE class has nodes as its instances. Each node has the following instance variables: the node-ID, the node-name, etc. the set of federations to which the node belongs, the database system (which includes the local policy, the schema, the DBMS, and the database), the administrator, local users, and global users. Administrator and users are instances of the PERSON class with instance variables which include Person-ID, Name, Type of user, and Nodes.

In addition to the instance variables shown in the figure, each node will have a database system as an instance variable. This instance variable is an object and represents the local database system. This system consists of the database, the local DBMS, the local schema, and other information such as the local security policy. That is, the database system object is a composite object. The specification of the component objects is yet to be defined. Note also that the local DBMSs may be relational systems or object-oriented systems. At a higher level of abstraction, we do not distinguish between these systems. That is, the interface to the database system object is uniform. The actual methods which implement the functions may be different for the various types of data models utilized.

We now discuss the management of the system with an example. This is illustrated in Figure 2. Suppose node N1 wants to join the federation F3. Node N1 sends a message to the class FEDERATION with federation F3 as a parameter (msg 1). Node N1 may also give some other information as to what information it needs from others and the information it is willing to share. When federation F3 gets the message, the corresponding method gets executed. The federated policy may be examined to see if N1 can join the federation (msg 2), where it is assumed that the administrator U3 maintains the federated policy.

By using this approach, the DOMS maintains autonomy, independence, and provides a natural and clear interface between object entities in the federation. This allows the federated environment to accommodate a wide variety of vendor products, DBMSs, schemas etc. which are abundant in today's heterogeneous environment. There is also a future possibility of gaining interoperability with non-database information systems.

At the same time, federation F3 may also send messages to the nodes which are already part of the federation to see if all of these nodes are willing for N1 to join the federation (msg 3 and msg 4). If all checks are satisfied, F3 sends a message to node N1 to join the federation and it includes N1 as part of the list of nodes which belong to it. That is, the value of the instance variable of F3 which specifies the nodes gets updated. Node N1 in turn updates the value of its instance variable for the federation.

Having shown how a database joins a federation, we discuss below some interoperations which can be achieved when a federated database is managed using this model.

3. INTEROPERABILITY IN A DOMS ENVIRONMENT

In this section we show how the object-oriented execution model above could also serve as the global data model in the federated environment. We will do this by explaining how some normal operations are achieved in such environment.

3.1 Overview

Normally, a user would interact mostly with his own database. However, in a federated database environment, a user can access a particular database, or the whole federation, subject to the policies which govern the federated environment.

An overview of our DOMS is shown in Figure 3. First, a mediator object acts as the main interface which receives from client objects (which contains a user's request), messages such as request to join or quit the federation, schema inspection, query, and so on. Similarly, it will also receive responses from other objects to which it has sent messages. In this way, the mediator object acts as the main 'broker' at
the federated level since it has to interact by sending more messages to other objects in the DOMS such as the schema object, query object, transaction object, security object and integrity object. The operations of these objects will be discussed in the following subsections. We will focus mainly on the query object and discuss only the essential points of the other objects.

![Figure 3 Overview of DOMS Environment](image)

### 3.2 Query Object

Although the distributed environment is modelled using an object model, due to the autonomy of the DBMSs, the users would often query the system in a language familiar to them. There are two levels of query possible in a federated database environment: local and federation. A query meant for the local level should be handled without much problem. This guarantees autonomy and saves users from the need to be trained to use other systems as a result of the database joining the federation. On the other hand, a query can be issued at the federation level, perhaps to gain a more comprehensive and up-to-date information.

Referring to Figure 4, suppose a user issues an SQL query. The client object should be able to distinguish whether the query is only at the local level or it is meant for the federated environment. If it is a federation level query, the interface which receives this query will treat it as an object and pass it to the mediator object. The mediator object will then decide which nodes will be sent further messages to fulfill the original global query.

![Figure 4 Query Processing in a DOMS Environment](image)

Now, if the query is a local one, then it is executed directly by the DBMS. On the other hand, it may be at the federation level in which case the target DBMS is a non-relational one such as a legacy DBMS. In this case the query has to be transformed into a language understood by the foreign DBMS. All the target DBMS objects which receive the query, will execute it, package the result and send it to the mediator object. The mediator object will in turn pack these results as a message and send it to the client object which has originally issued the global query.

Furthermore, suppose the federated environment provides facilities for query optimization via an optimizer object as shown in Figure 4. Thus a query request can be optimized by passing them to the object first (shown by dotted arrows), before actually sending it to the target database object. This is particularly true for the translated queries, since such versions would most probably contain many redundancies. An optimizer object when present will also benefit new users who are not yet able to issue a compact optimum query. In a federated environment which spans across a network over long distances, optimization will also reduce cost and response time. Finally, an expert user may choose to ignore the optimization facility altogether, and this should be handled easily by the DOMS.

### 3.3 Transaction Object

Transaction management with distributed updates is still a difficult problem for heterogeneous database systems. A transaction is a program unit that must be executed in its entirety or not executed at all. In a distributed environment, a transaction may have many subtransactions executing at different sites.

The DOMS environment provides a modular approach to transaction management by using a transaction processing object as shown in Figure 5. The mediator object gives a user's request to the transaction object. This object has information about the transaction, communicates with the global scheduler, which is also an object, and gets the execution strategy. Then it communicates with the local transaction managers (which are part of the local DBMSs) possibly through the mediator and translator objects.

![Figure 5 Transaction Management in a DOMS Environment](image)

### 3.4 Security Object

Suppose a user issues a request at the global level, and the mediator object then identifies the databases which have to be interrogated in order to answer the query. To access the identified databases, the user needs to have enough security clearance. This is particularly necessary, when the databases do not belong to him or his department. To ascertain this, the mediator object will first send to the security object a security-check message containing parameters such as the user’s id, his current clearance level and the node-ids for these databases. The security object in turn will send a response message, with parameters which indicate accessibility to the required databases.

It is not difficult to see the clean separation of processing between objects in DOMS. In this example, the mediator object, after receiving the response message from the security object, may decide to hide some
databases, since the user who issues the queries might not have access to them (see figure 3). This action may be necessary for the purposes of the system because the mediator object accepts query messages, which of course is accessible by this user.

One major issue on integrating heterogeneous databases is the enforcement of integrity constraints on objects. For example, one DBMS may enforce a minimum of 20 hours whilst another may enforce a minimum of 35 hours. If there is a query to retrieve all objects that work a minimum of 25 hours and subsequently give it to the query context based on whether the request is a query

The SCMS Manager is responsible for managing the schemas of the DBMSes. It has access to and communicates with the local DBMSes via their schemas. Furthermore, it is the SCMS Manager's responsibility to perform schema integration and provide services such as query, transaction, integrity and concurrency control. The schema object to obtain the schemas to retrieve the data from the local DBMS and subsequently give it to the user. The general role of the schema object is to integrate heterogeneous databases.

![Diagram of Management in a DOMS Environment]

The research in federated database environment is to provide a feasible way of managing its multiplicities of data. It has been shown that a feasible way to handle the overall environment is the object oriented approach. The approach possesses the advantages of independence, expressive power, and a natural and intuitive way of thinking for the user.

The advantages of this approach is its ability to reuse existing software as the component objects of the system. The SCMS Manager is conducting a more detailed investigation in the usage of components using the DOMS approach. We will see the key to managing future large scale databases. The SCMS Manager is conducting a more detailed investigation in the usage of components using the DOMS approach.

REFERENCES


