Design, Share and Re-use of Data and Applications into a Federate DataBase System

Thierry Millan, Myriam Lamolle, Frédéric Mulatero

Design, Share and Re-use of Data and Applications into a Federate DataBase System

Thierry MILLAN, Myriam LAMOLLE
Team of Pierre BAZEX
IRIT - CNRS (UMR 5055) - Université Paul Sabatier
118, route de Narbonne
31062, Toulouse Cedex FRANCE
Tél. (+33) (0)5 61 55 86 32 - Fax. (+33) (0)5 61 55 62 58
E-mail: (millan, lamolle)@irit.fr

Frédéric MULATERO
MIRA – Université Toulouse Le Mirail
Département de Mathématiques
5, allée Antonio MACHADO
31058 TOULOUSE Cedex
Tél. (+33) (0)5 61 50 42 20
E-mail: mulatero@univ-tlse2.fr

Abstract: This paper presents an architecture to federate preexisting hospital databases. This solution relies on an oriented-object approach, persistence by reachability and a new MultiDataBase System (MDBS) architecture. The MDBS allows to develop global applications to exploit all the DataBase Management System (DBMS) resources distributed across the hospital network. However, the emergency of global applications should not disturb the preexisting local application. Furthermore, the MDBS provides data accessibility for preexisting applications.

The MDBS is in charge to manage global transactions and to collect global data when no more used.

Keywords: persistence by reachability, MultiDataBase System, oriented-object approach, re-use

1. Introduction

A new trend is to federate preexisting and independent DBMS into MDBS. In France, medical information system are made up of heterogeneous applications. Each application is specific to a medical service. Indeed, radiological services store and handle numeric pictures. Hematological services store and handle results of hematological analysis results. In addition, each service stores and handles redundant administrative information about their patients.
This study aims at providing access to information stored in the hospital databases. Furthermore, the following constraints must be taking under account:

- data accessibility from preexisting applications;
- integration of the preexisting applications: usual users must not be disturbed by this integration;
- the emergency of global applications should not disturb the preexisting local application.

Some studies have been carried out yet. In the MISA project [Gan96], the federation of the existing databases is done using an Object-Oriented DBMS. This solution requires a new DataBase Management System (DBMS).

Our solution avoids to use an another DBMS and to define a new global schema. This solution relies on a new MDBS architecture. A MDBS [ÖV91] provides uniform access to data managed by DBMS that can be heterogeneous and that are distributed across a network.
In our approach it is fundamental to preserve the autonomy of the federated DBMS (preexisting applications, concurrency control, garbage collector, ...). However, the emergency of global applications should not disturb the preexisting local application running on each local DBMS. This solution aims at respecting DBMS autonomy.

2. Applications and objects

Two types of applications coexist in a federated MDBS. *Local applications* are limited to the manipulation of local objects stored in one DBMS. These manipulations escape to the control of the MDBS. *Global applications* transparently access to shared data located in the different DBMS federated by the MDBS.
For each DBMS involved in the federation, objects are partitioned in three disjoint sets (see Figure 2):

- *local objects* are only accessed by *local applications*. References between *local objects* are local to one DBMS. For example, during the radiology report redaction, this report is local to the radiology service. The redaction is done using a *local application*.
- *shared local objects* are accessed both by local and global applications. Such objects are *local objects* that are made visible to the *global applications*. They do not contain external references. Thus *local application* not under the control of the MDBS can not reach a *global object* through a *shared local object*. For example, when the radiology report redaction is achieved, the writer puts it into the corresponding patient's folder. This folder is a *shared local object* accessible by the other hospital's services ;
- *global objects* are shared by *global applications* under the control of the MDBS. A *global object* may contain external references, that are references to *global objects* located in remote DBMS. For example, all the reports could be put into a global set, which is a *global object*, accessible from others hospitals or by family doctors via an Internet access.

Applying referential persistency [ABC+83], objects are said to be persistent if they are named or referenced through a named object. In order to preserve DBMS autonomy, each DBMS manages its own set of *local names* for persistency roots of local objects. The MDBS manages a unique set of *global names* for persistency roots of global objects as pictured in Figure 2.
As mentioned above, two types of reference coexist in the federated MDBS:

- *Local references* are used to identify or send operations to local and shared local objects inside a local DBMS;
- *External references* are necessary to access global and shared local objects in the federated MDBS context.

While local references provide direct access to objects inside a DBMS, external reference provide indirect access to objects across the network. As in [LQP92], an external references to object *o* is composed of (see Figure 1) a local reference to an *exit item* stored in the local DBMS, which in turn references an *entry item* stored in the DBMS containing object *o*, which itself contains a local reference to object *o*. The reference between an entry item and an *exit item* is a *remote reference* that can be implemented as the concatenation of a DBMS identifier and a local reference.

3. General Architecture

There is no restriction in combining global objects to construct complex objects. Figure 1 shows an example where global object *o1* stored in *DBMS1* is composed of global object *o2* stored in *DBMS2* in turn composed of object *o3* stored in *DBMS3* in turn composed of object *o4* stored in *DBMS1*.

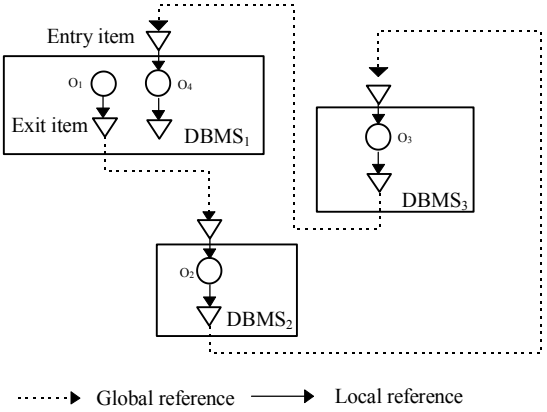
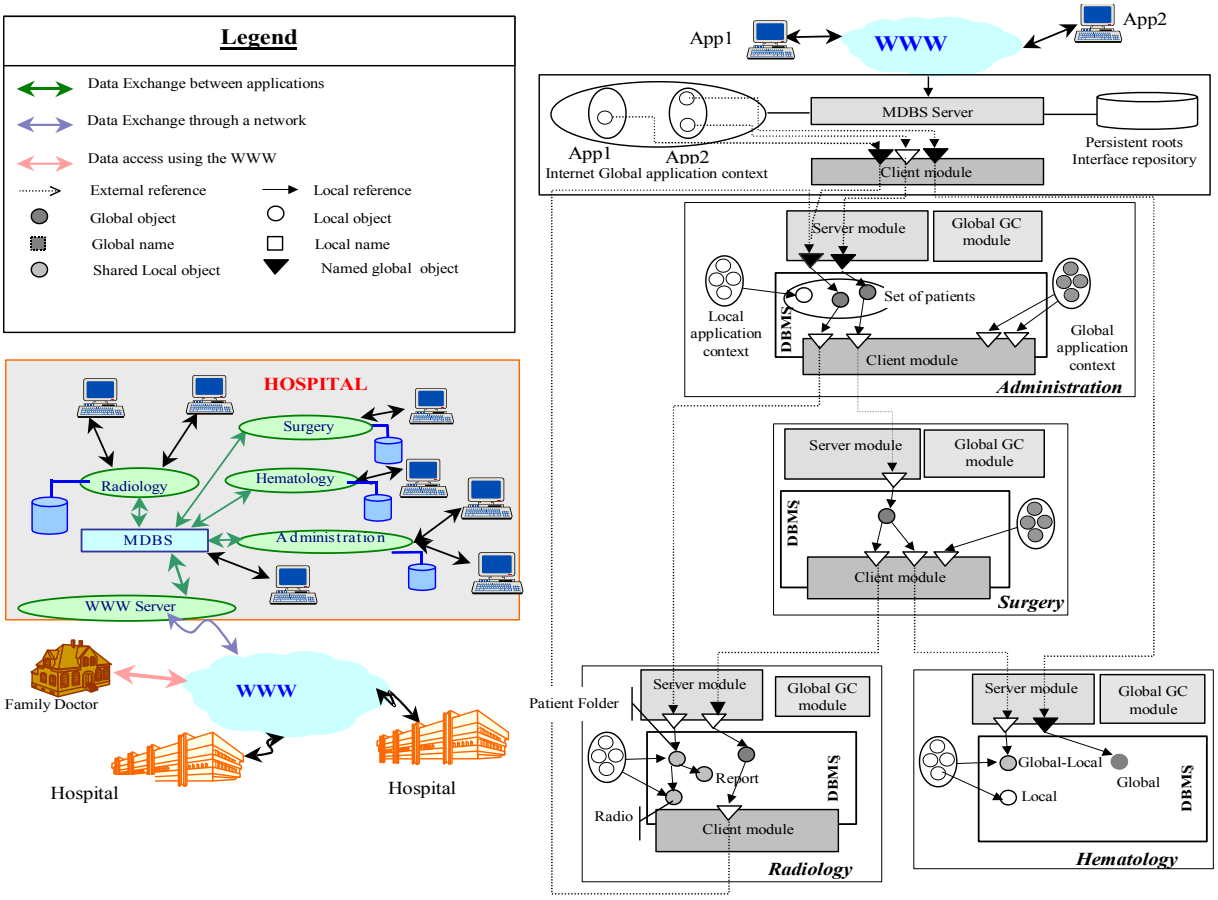


Figure 1: Combining Global Objects

This example shows that there is no limitation in the length of the composition string of complex objects and that there is no hierarchy between the DBMS involved in the composition. As a consequence, the same DBMS can in turn play the role of a client or a server. It is considered as a *client* when it calls the interface of a global object stored in a remote DBMS, while it is considered as a *server* when it implements the interface of a global object. In order to integrate a new DBMS into the federation, two modules have to be developed, namely the DBMS client module and the DBMS server module, so that the DBMS can play its two roles (see Figure 2 for an example).



a- General Architecture of a Hospital information system

***b*- General MDBS Architecture**

Figure 2: General Architecture

3.1. DBMS client module

The client module is in charge to provide both the MDBS interface to global applications and the global object interface. The MDBS interface is composed of the following operations: connection to the federated MDBS, disconnection, commit of a transaction and abort of a transaction. The *connect* operation initiates a first transaction for the global application and the *disconnect* operation commit the last transaction of the application. The *commit* operation must ensure the durability of all the operations performed by the transaction (Atomicity and Durability) and must follow a two step commit protocol since we are in a distributed context. The *abort* operation must ensure that all the operations of a transaction are undone (Atomicity). The transactional aspects of this interface are further discussed in section 3.4.

The global object interface consists in sending global object method calls to the appropriate DBMS server module as well as to provide access to the list of global names. This can be done following CORBA [OMG91] by mean of proxy whose objective is to encapsulate object method calls through *exit items*. It is important to mention here that since we are in a transactional context, each global object operation is performed for one transaction whose identifier is transmitted somehow. The simplest way to do it is to add the transaction identifier as a parameter of the method call mechanism.

Exit items are managed by the client module. *Exit items* are implemented as a class of local objects stored in the DBMS associated to the client module so that they can be referenced through a local reference. If the *exit item* is already assigned to a global object, the client item creates a new *exit item* so that there is one exit item per external references held in the DBMS even if they refer to the same object. Having several *exit items* pointing to the same object avoids contention on *exit items* due to the DBMS transaction management mechanism (an exit item updated by one transaction is locked for the whole transaction.

3.2. DBMS server module

The server module provides the mirror interfaces of the client module interfaces on the server side. These interfaces provide access to the transactional facilities mentioned above and transmit global objects methods calls to the appropriate object stored in a local DBMS through an entry item. This can be done by mean of stubs like in CORBA.

Entry items are managed by the server module. Contrarily to *exit items*, entry items are not stored in the local DBMS. In this way it is easier to provide uniform remote references in each server module. Another advantage of implementing entry items outside DBMS is to ensure that coherent parallel updates can be performed on entry items by mean of *latches* [ML89] instead of transactions. *Latches* are instant duration locks that guarantee the coherency property of parallel updates without providing the atomicity property of transactions.

3.3. About Transaction Management

The specificity of database objects is that they have to respect the ACID properties [GR93], namely: Atomicity, Coherency, Isolation and Durability. Global objects as well as local objects must satisfy ACID properties. Consequently, database objects are manipulated through transactions.

The MDBS distinguishes global transactions accessing global and shared local objects from local transactions executed on each DBMS. A global transaction is supported by one local transaction per DBMS involved in the global transaction. As a global transaction commits, the commitment of the involved local transactions is synchronized though a two phase commit protocol [GR93]. Commercial DBMS provide a standard interface for the two phase commit protocol. This interface is encapsulated by the DBMS client and server modules.

3.4. Garbage Collection [MTB98a] [MTB98b]

The purpose of the global GC is to make sure that entry items, global object cells and *exit items* are destroyed when no more used. On the other side, the purpose of a local GC is to collect the

local objects no more used in a DBMS. The global GC is a distributed task and implemented by modules associated to each DBMS.
Each component DBMS is supposed to have its own local GC and none assumption is made on the behavior of the local GC. It has the following interesting properties:

- there is no interaction between the global GC and the local GC;
- it is incremental;
- it requires few interactions with transactions;
- it is able to detect dead object cycles that are frequent in a DBMS context;
- it is able to collect objects without accessing the whole database;
- and global synchronization of DBMS sites is not required;
- The Global GC works exclusively on entry and *exit items* without accessing global object cells stored in DBMS. Consequently it implies few I/O overhead for the DBMS.

4. Conclusion

This paper presents a concrete example of the MDBS use. We show how to federate existing hospital DBMS without any modification of the local applications.
We consider data as objects with three visibility levels according to their use (local, shared local and global objects). The MDBS satisfies the constraints expressed in the introduction. The main difference with other approaches (MISA [Gan96]) is that this solution does not need an external database as common base to federate the local databases. Consequently, this solution presents the best cost-effectiveness ratio thanks to the re-use of the preexisting databases, software and hardware, and the minimization of the training times.

Currently, we are carrying out studies concerning the development of persistent application using heterogeneous languages (Ada, Java[MM98]). Such bindings could offer valuable solutions to develop global applications that exploit global objects (Internet applications using Java for example). Furthermore, they could also make easier the migration of local applications to global applications.

References

[ABC+83] M. Atkinson, P. Bailey, K. Chisholm, P. Cockshott, R. Morrison, *An Approach to Persistent Programming,* Computer Journal, 26(4), 1983

[Gan96] *B. Gandner, MISA : Medical Information System Architecture – Application Multibase*, Mémoire présenté en vue d'obtenir le diplôme d'ingénieur CNAM en Informatique, June 1996

[GR93] J. Gray, A. Reuter, *Transaction Processing: Concepts and Techniques,* Morgan Kaufmann, 1993.

[LQP92] B. Lang, C. Queinnec, J. Piquer, *Garbage Collecting the World,* Conf. Record of the Nineteenth Annual ACM Symposium of Principles of Programming Languages, 1992

[MM98] T. Millan, F. Mulatéro, *coupling between languages and Object Oriented Database Management Systems to exchange data between applications developed using heterogeneous languages,* Expersys'98, Virginia, United-States of America, November 1998

[MTB98a] F. Mulatero, J.-M. Thevenin, P. Bazex, *Multi-Base Garbage Collector Algorithms,* BDA'98, Hammamet, Tunisia October 1998

[MTB98b] F. Mulatero, J.-M. Thevenin, P. Bazex, *A global Garbage Collector for Federate Database Management Systems*, Dexa 98, Vienna, Autria, September 1998

[OMG91] Object Management Group, *The Common Object Request Broker Architecture : Architecture and Spec.,* OMG Document Number 91.12.1 Revision 1.1, 1991

[ÖV91] M. T. Özsu, P. Valduriez, *Principles of Distributed Database Systems,* Prentice-Hall Int. Editions, 1991