# Design and Implementation of a Component-based Concurrency Control Mechanism for a Distributed Database

**Maximiliano Canché[1], Juan Lavariega[2] and Erika Llanes[3]**

**[1] Faculty of Mathematics, Autonomous University of Yucatan**
**Tizimín, Yucatán, México**

**[2] Department of Computer Science, Monterrey Technological Institute**
**Monterrey, Nuevo León, México**

**[3] Faculty of Mathematics, Autonomous University of Yucatan**
**Tizimín, Yucatán, México**

**Abstract**

Currently, component-based systems offer substantial benefits due to their ability to divide responsibilities. Some of the main benefits of developing components as independent parts of a system are: reduced costs, reduced implementation effort and the addition or replacement of modular functions. Moreover, since there is no well-defined architecture for distributed database systems, commercial companies apply the concept of distribution differently. One option for getting the functionality of a Distributed Database Management System is extending a monolithic Database Management System of fragmented way, i.e. adding or replacing functions in a modular form. In this paper, we design and develop a concurrency control component in order to synchronize access to data in a distributed database and describe the advantages of implementing it as an extension of a monolithic Database Management System. The component is developed with Open Source tools because these elements allow developers to further improve these elements with relative ease.

***Keywords:*** *Software Components, Component-based, Distributed Database, Concurrency Control.*

## 1. Introduction

The complexity and resource requirements of today's software systems create the need for flexibility, adaptability, and ease of composition or reuse. Component-based software development can be considered as an evolutionary step beyond object-oriented development in achieving these goals [1]. This can reduce build time, because instead of developing a complete system we can adapt an existing one to our needs by adding the required functionality. With this approach, it is possible to modify the behavior of a system by adding, removing or exchanging components.

One technology that helps us in building component-based applications facilitating adaptability and independence are Open Source Systems [2]. With these tools it is possible to integrate devices from various sources on a system and obtain a rapid adoption of the new technology thanks to the existing competition. Furthermore, the costs of open technologies are considerably lower compared to proprietary technologies.

Distributed data processing is a need that nowadays many organizations try to satisfy. In a Distributed Database (DDB) [3] transactions are executed which should be coordinated by Database Management Systems (DBMS's), particularly by the transaction manager, in order to maintain desirable properties for the effective operation of your organization.

Concurrent access in transactions is essential because of the nature of the information and of the multiple users that wish to access it at the same time. When this occurs it is necessary for the DBMS to ensure transactions serialization, i.e. the result should be the same as if they were running sequentially in a specific order. Concurrency control becomes more relevant when the database operates in a distributed environment. This requires a distributed synchronization algorithm which must ensure that concurrent transactions are not only serialized at the point where they are running, but also serialized globally (implying that the order in which they are executed at each point should be identical). The Concurrency Control mechanisms must maintain both the consistency and isolation properties of transactions, hence the importance of concurrency control as a key component of these [3].

A desirable feature for the architectures of the DBMS's is the possibility of defining them in fragments (components) such that new components can be added or existing components can be exchanged in a flexible manner [9]. With this approach it is possible to adapt distributed functions to the behavior of a DBMS.

The aim of this paper is to provide a framework which provides extensibility to a monolithic DBMS creating a concurrency control mechanism based on prefabricated software components in order to obtain the necessary functionality to synchronize access to the elements of a Distributed Database and maintain the consistency thereof. This mechanism runs with the primary support of Open Source Systems (being built and adapted using public domain tools), to guide the evolution of DDB in such systems.

## 2. Methodology

We took as a base the architecture of Concurrency Control for multiuser applications described in [5]. This architecture allows the integration of concurrency control in different environments, based on the concept of systems building with independent prefabricated parts.

Additionally we considered the classification of mechanisms defined in [3]. Next we adapted the architecture to a distributed environment to design and develop the Concurrency Control component using the strict two-phase locking technique (2PL strict) [4], and finally we conducted tests and analysis of its performance in an Open Source environment.

## 3. Design of the component

### 3.1 Component-based distributed architecture

A component architecture is a description of a system in form of a collection of components that interact with each other through connectors, these are structural relationships and behavioral dependencies. In other words, the architecture defines a standard for the composition of components into a system in terms of what makes a component, how are interfaces described and how do components interact and communicate [1]. In this paper we define a four-layer, component-based architecture for the interaction of components in a distributed environment, which is shown in Figure 1.

In the top layer of the architecture we considered the execution of user applications which request transaction operations. For the next layer we defined the operating environment of the Distributed Transaction Manager called TransactionManager, which interacts with the locking component denominated LockManager -an essential part of this work- and the Global Database (GlobalDB) which maintain the lock units. Atomic units to be held in transactions are identified and planned by TransactionManager. In the third layer are the agents [6] of various local sites found in the Distributed database, which are actually component processes running and waiting for requests. Finally, in the fourth layer we have the Database Management Systems (DBMS) at their respective points accessing their local data.

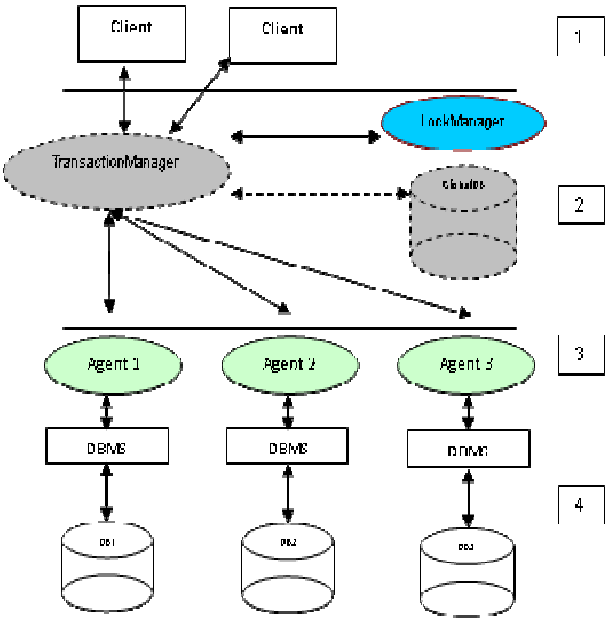


Fig. 1. Architecture and distributed environment of components

### 3.2 Components of concurrency control and its functions

In the described distributed architecture, the component LockManager performs the basic concurrency control functions. It is responsible for maintaining the locks in a database and is designed to be easy to use and to extend to other features of a DBMS. This component provides the following services to a transaction manager:

- Receiving data lock requests with three levels of granularity: field, record, and table levels.

- Granting (or denying when necessary) lock requests based on verification of conflicts for its decision. If the application is for a lock on a data such that there is no conflict on this lock, the lock is granted, otherwise it is denied.

- Receiving requests for the release of a particular lock given its lock number, which is unique.

- Receiving requests for the reléase of all locks of a transaction.

- Keeping the transactions synchronized in order to provide them with serialization.

To provide these services the Concurrency Control component is based on the following set of elements:

- LockObject: A component that serves as a common language between LockManager and the transaction manager. It contains items to be blocked and a unique ID for each.

- Restriction: A component used by LockObject and LockManager. It maintains a structure that stores the restrictions (conditions) of a lock object.

- Dblock: A database consisting of two basic relationships: the relationship of locks and the relationship of constraints.

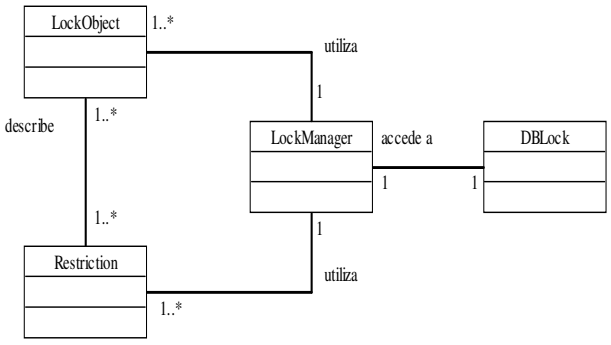


Fig. 2. Interaction of the component's elements

### 3.3 Functionality

Transactions must be received by a transaction manager which will create a list of objects related to lock and send it to LockManager through a request to the respective locking. The requests made by the transaction manager to obtain a lock are concurrent processes (independent of each other), which will be competing for LockManager. Hence the need to maintain synchronization of LockManager in requests. This is shown in Figure 3.

Additionally, each site must have an agent which is an application server to run requests of the transaction manager. Each agent is a process that interacts with the DBMS of the local site for the purposes of transactions in the distributed database.

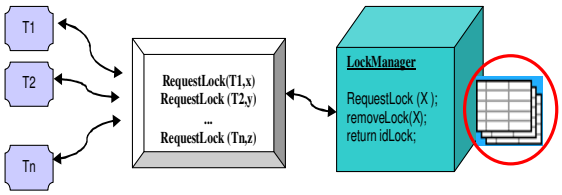


Fig. 3. Transactions competing for LockManager

## 4. Implementation and testing

The component implementation was done in Java Language. Java is a precompiled language that provides ease of use of tools such as database access, extensive libraries of data structures, methods of remote invocation and object access synchronization [7]. Java provides mechanisms to synchronize access to methods when they are accessed concurrently. Java associates a lock with each object that has synchronized code when it is accessed and releases it when the object is left. This is done automatically and atomically by Java runtime system, which frees the developer from the task of implementing locking mechanisms to methods [8]. The greatest strength of Java is its mobility in different environments. The logic implemented in Java can be moved from site to site in a distributed system using both hardware and heterogeneous operating systems. Achieving this in a consistent manner is a goal of components implemented in Java, hence the choice of this language for the implementation of this concurrency control component.

We used the MySQL DBMS [10] at each point and the Linux Operating System as a development environment. We also used the Windows XP Operating System for client applications and for testing the multiplatform link with remote method invocations between the two operating systems.

We performed a funds transfer application as a test scenario. A database was distributed at two points of the network and the fragmentation criteria were the following:

- Site 1: Records with [No_Cuenta <50]
- Site 2: Records with [No_Cuenta> = 50]

We developed a prototype for the transaction manager (TransactionManager) which received requests from customers. This was in charge of building the lock objects of LockManager (the main component of this work).

The client interface was performed using the package javax.swing for the access to TransactionManager. The interface receives data for two pairs of accounts (Source, and Destination) and the amounts to be transferred between accounts. Both the application interface and the source code for the creation of a lock object in LockManager are shown in Figures 4 and 5 respectively.

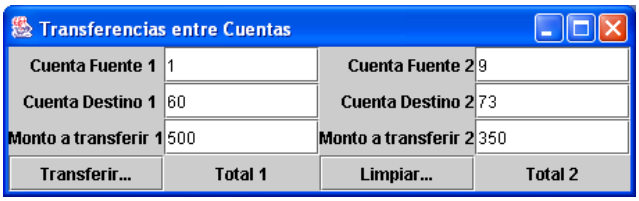


Fig. 4. Account Transfer Application

```
LockObject lockObject = new
LockObject(idTransaction, lockType, globalTable,
restrictions);

int idLock;
int notLocking = 0;
do {
    idLock    =    LM.requestLock    (lockObject);
//Critical section of LockManager
    if (idLock == 0) {
       try {
            Thread.sleep(150);
            notLocking++;
       }
       catch (Exception e) {
            System.out.println(e.getMessage());
       }
    }
 } while (idLock == 0 && notLocking <5);
```

Figure 5. Code of the lock object creation in LockManager

## 5. Conclusions and future work

In this paper, we present a study which aims to continue applying the approach of massive exploitation of Open Source resources in developing configurable and adaptable components to different DBMS's. The main advantages of this development approach are:

- Applications can be easily adapted to changing components (for example diverse DBMS's).
- The use of these kinds of components provides desirable characteristics in systems such as scalability, reuse, and encapsulation.

We consider that the use of a scheme like the one used in this paper facilitates the construction of software and system extensibility with less cost and effort reducing the possibility of errors in applications.

Based on the experience gained in the development of this work we have considered future work on the extension of the Lock Manager to a physically distributed Lock Manager. Since the component is administered only on a centralized server, we will have problems if the server fails. A key area of research is the extension of functionality to a physically distributed environment.

### Acknowledgments

The authors would like to acknowledge Autonomous University of Yucatan for providing the facilities to finish this work.

## References

[1] C. Atkinson and C. Bunse, Component-Based Software Development for Embedded Systems, Springer, 2005.
[2] Open Source Initiative, http://opensource.org/, (accessed August 12, 2012).
[3] M.T. Ozsu and Patrick Valduriez. Principles of Distributed Database Systems, Springer, 2011.
[4] T. Sultan, H. El-Bakry and H. A. Hameed, Design of Efficient Dynamic Replica Control Algorithm for Periodic/Aperiodic Transactions in Distributed Real-Time Databases, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 2, January 2012.
[5] G. Heineman, An Architecture for Integrating Concurrency Control into Environment Frameworks, 17th International Conference on Software Engineering, Seattle, WA, 1995.
[6] S. Ceri and G. Pelagatti, Distributed Databases, Principles & Systems, Mc. Graw Hill, 1985.
[7] Y. Bai, Practical Database Programming with Java. Wiley-IEEE Press, 2011.

[8] B. Goetz and T. Peierls, Java Concurrency in Practice, Addison-Wesley Professional, 2006.
[9] K. Dittrich and A. Geppert, Component Database Systems, The Morgan Kaufmann Publishers, 2000.
[10] R. Bradford, Effective MySQL Optimizing SQL Statements (Oracle Press), McGraw-Hill Osborne Media, 2011.

**Maximiliano Canché-Euán** obtained his degree in Computer Science from the Autonomous University of Yucatan, Mexico, in 2000 and his M. Sc. Degree in Information Technology from Monterrey Technological Institute (ITESM) in 2002. He is a professor of Computer Science at the Faculty of Mathematics at the Autonomous University of Yucatan. Currently he is giving courses on programming of mobile devices, databases and computer animation in professional programs at the UADY. His research lines are: Animation, Networks, Programming and Databases.

**Juan Carlos Lavariega-Jarquín** obtained his Bachelor degree and his M.Sc. degree in Computer Science from Monterrey Technological Institute (ITESM) in 1987 and 1990 respectively. He obtained his phD degree in Computer Science from Arizona State University in 1999. Currently he is a full-time lecturer at Monterrey Technological Institute, Campus Monterrey. He is a member of the IEEE Computer Society. His specialization areas are Database Systems, Software Engineering and Operating Systems.

**Erika Rossana Llanes-Castro** received her degree in Computer Science from the Autonomous University of Yucatan (UADY) in 2002 and her M. Sc.Degree in Computer Science from Monterrey Technological Institute (ITESM), Campus Estado de Mexico in 2011. Currently, she is a full time academic technician at the Autonomous University of Yucatan since 2002 in the department of Computer Science in Tizimín México. She has participated in software engineering development projects. Currently is giving courses on programming mobile devices in the professional programs at the UADY.