Overview of the Benefits and Costs of Integrating Heterogeneous Applications by Using Relaxed ACID Properties

Lars Frank
Department of IT Management
Copenhagen Business School, Howitzvej 60, Frederiksberg, Denmark
+45 38 15 2400, DK – 2000, lf.itm@cbs.dk

ABSTRACT

In central databases the consistency of data is normally implemented by using the ACID (Atomicity, Consistency, Isolation and Durability) properties of a DBMS (Data Base Management System). In practice, it is not possible to implement the ACID properties if heterogeneous or distributed databases are involved and the availability of data also has to be optimized. The objective of relaxed ACID properties across different database systems is that the users can trust the data they use even if the databases involved are temporarily inconsistent. Therefore, we will in this paper use relaxed ACID properties across different database systems or ERP (Enterprise Resource Planning) modules in order to make it possible to select the best set of modules even though the combination of modules may be heterogeneous. That is, it should be possible to select the best of breed software for the different business functions of the company. The objective of this paper is to give an overview of both the benefits and costs of using application/database integration with relaxed ACID properties.

Keywords

Relaxed ACID properties, Integrated databases, Integrated ERP modules, Heterogeneous databases

1. INTRODUCTION

When using a DBMS to manage the database an important aspect involves transactions which are any logical operation on data and per definition database transactions must be atomic, consistent, isolated, and durable in order for the transaction to be reliable and coherent.

The ACID properties of a database are delivered by a DBMS to make database recovery easier and make it possible in a multi user environment to give concurrent transactions a consistent chronological view of the data in the database. The ACID properties are consequently important for users who need a consistent view of the data in a database. Information systems that use different DBMS systems can be integrated by using more or less common data and/or by exchanging information between the systems involved. In both situations, the union of the databases of the different systems may be implemented as a database with so called relaxed ACID properties where temporary inconsistencies may occur in a controlled manner. However, when implementing relaxed ACID properties it is important that from a user's point of view it must still seem as if traditional ACID properties were implemented, which therefore will keep the local databases trustworthy for decision making.

1.1 Relaxed ACID properties

The Atomicity property of a DBMS guarantees that either all the updates of a transaction are committed/executed or no updates are committed/executed. This property makes it possible to re-execute a transaction that has failed after execution of some of its updates. In distributed databases, this property is especially important if data are replicated as inconsistency will occur if only a subset of data is replicated. The Atomicity property of a DBMS is implemented by using a DBMS log file with all the database changes made by the transactions. The global Atomicity property of databases with relaxed ACID properties is implemented by using compensatable, pivot and retriable subtransactions in that order.. By applying these subtransactions it is allowed to commit/execute only part of the transaction and still consider the transaction to be atomic as the data converge towards a consistent state.

The global Consistency property is not defined in databases with relaxed ACID properties because normally such databases are inconsistent and this inconsistency may be managed in the same way as the relaxed Isolation property.

The Isolation property of a DBMS guarantees that the updates of a transaction cannot be seen by other concurrent transactions until the transaction is committed/executed. That is the inconsistencies caused by a transaction that has not executed all its updates cannot be seen by other transactions. The Isolation property of a DBMS may be implemented by locking all records used by a transaction. That is the locked records cannot be used by other transactions before the locks are

released when the transaction is committed. The global Isolation property of databases with relaxed ACID properties is implemented by using countermeasures against the inconsistencies/anomalies that may occur.

The Durability property of a DBMS guarantees that the updates of a transaction cannot be lost if the transaction is committed. The Durability property of a DBMS is implemented by using a DBMS log file with all the database changes made by the transactions. By restoring the updates of the committed transactions it is possible to recover a database even in case if it is destroyed. The global Durability property of databases with relaxed ACID properties is implemented by using the local Durability property of the local databases involved.

A *heterogeneous modular ERP architecture* is an integrated set of ERP systems in different locations where each ERP module can use the resources of the other ERP modules in its own or in other ERP locations. In a heterogeneous modular ERP architecture, the ACID properties must be relaxed both across locations and across modules in the same location. That is within a module the traditional ACID properties of a DBMS may be used while relaxed ACID properties are implemented by the modules themselves for all accesses/updates across modules. Each ERP module has full autonomy over its own resources/tables. That is, its tables may be recovered independently from the tables owned by other ERP modules.

1.2 Objective

As described earlier the objective of this paper is to give an overview of both the benefits and costs of using the integration architecture described in this paper.

The overriding benefit of using relaxed ACID properties across different application modules/databases is flexibility. This flexibility may be described as flexibility to integrate different systems/modules, Flexibility to migrate systems/modules without using a risky overnight conversion, or the flexibility to distribute applications/modules to the locations where they are needed. Distribution may also be used to increase the total capacity of a system.

Besides flexibility it is also possible to achieve increased availability as this was the original reason for introducing relaxed ACID properties [14]. In this paper, the analyses of costs are divided into development, implementation, and operative costs.

The paper is organized as follows: First, the transaction model used is described. In section 3, the properties of the heterogeneous modular ERP architecture that is necessary for integrating heterogeneous ERP modules is described. In section 4, we illustrate by examples how it is possible achieve the benefits of using integrated heterogeneous ERP modules. Section 5 analyses the different types of costs that are introduced by using our heterogeneous modular ERP architecture. Finally, we have conclusions and future work.

Related Research: The transaction model used in this paper is *the countermeasure transaction model* [6]. This model uses the relaxed Atomicity property as developed in [14, 19 and 20]. How to use countermeasures against consistency anomalies are described in [9]. The model also uses countermeasures against the isolation anomalies. These anomalies are described in [1 and 2]. How to use countermeasures against isolation anomalies was first systematized in [12]. In the countermeasure transaction model the problem of implementing the Durability property is solved as describe in [2]. We might call our transactions as '*flexible transactions'* [20] as the only new facility is the use of countermeasures to manage the missing consistency and isolation properties. Many researchers [16 and 17 have dealt with Integration patterns/architectures. However, to our knowledge they only take the atomicity property into account and do not deal with the ACID properties in general when they are more or less missing in the period of migration. The method to use integration architecture for flexible migrating from one ERP system to another has been described in [5]. The method to integrate heterogeneous distributed databases in general by using SOA services is described in [8 and 10].

2. THE TRANSACTIONAL MODEL

A *multidatabase* is a union of local autonomous databases. *Global transactions* access data located in more than one local database. In recent years, many transaction models have been designed to integrate local databases without using a distributed DBMS. The countermeasure transaction model [12] has, among other things, selected and integrated properties from these transaction models to reduce the problems caused by the missing ACID properties in a distributed database that is not managed by a distributed DBMS. In the countermeasure transaction model, a global transaction involves a *root transaction* (client transaction) and several single site *subtransactions* (server transactions). Subtransactions may be nested transactions, i.e. a subtransaction may be a *parent transaction* for other subtransactions. All communication with the user is managed from the root transaction, and all data is accessed through subtransactions. The following subsections will give a broad outline of how relaxed ACID properties are implemented.

2.1 The Atomicity Property

An updating transaction has the *atomicity property* and is called *atomic* if either all or none of its updates are executed. In the countermeasure transaction model, the

global transaction is partitioned into the following types of subtransactions executed in different locations:

The *pivot* subtransaction that manages the atomicity of the global transaction. The global transaction is committed when the pivot subtransaction is committed locally. If the pivot subtransaction aborts, all the updates of the other subtransactions must be compensated.

The *compensatable* subtransactions that all may be compensated. Compensatable subtransactions must always be executed before the pivot subtransaction is executed to make it possible to compensate them if the pivot subtransaction cannot be committed. A compensatable subtransaction may be compensated by executing a *compensating* subtransaction.

The *retriable* subtransactions that are designed in such a way that the execution is guaranteed to commit locally (sooner or later) if the pivot subtransaction has been committed.

As described earlier, the global atomicity property is implemented by executing the compensatable, pivot and retriable subtransactions of a global transaction in that order. For example, if the global transaction fails before the pivot has been committed, it is possible to remove the updates of the global transaction by compensation. If the global transaction fails after the pivot has been committed, the remaining retriable subtransactions will be (re)executed automatically until all the updates of the global transaction have been committed.

2.2 The Consistency Property

A database is *consistent* if its data complies with the consistency rules of the database. If the database is consistent both when a transaction starts and when it has been completed and committed, the execution has the *consistency property*. Transaction *consistency rules* may be implemented as a control program that rejects the commitment of transactions, which do not comply with the consistency rules.

The above definition of the consistency property is not useful in distributed databases with relaxed ACID properties because such a database is almost always inconsistent. However, a distributed database with relaxed ACID properties should have *asymptotic consistency*, i.e. the database should converge towards a consistent state when all active transactions have been committed/compensated. Therefore, the following property is essential in distributed databases with relaxed ACID properties:

> If the database is asymptotically consistent when a transaction starts and also when it has been committed, the execution has the *relaxed consistency property*.

2.3 The Isolation Property

The isolation property is normally implemented by using *long duration locks*, which are locks that are held until the global transaction has been committed (Frank and Zahle, 1998). In the countermeasure transaction model, long duration locks cannot instigate isolated global execution as retriable subtransactions may be executed after the global transaction has been committed in the pivot location. Therefore, *short duration locks* are used, i.e. locks that are released immediately after a subtransaction has been committed/aborted locally. To ensure high availability in locked data, short duration locks should also be used in compensatable subtransactions, just as locks should be released before interaction with a user. This is not a problem in the countermeasure transaction model as the traditional isolation property in retriable subtransactions is lost anyway. If only short duration locks are used, it is impossible to block data. (Data is *blocked* if it is locked by a subtransaction that loses the connection to the "coordinator" (the pivot subtransaction) managing the global commit/abort decision). When transactions are executed without isolation, the so-called *isolation anomalies* may occur. In the countermeasure transaction model, relaxed isolation can be implemented by using countermeasures against the isolation anomalies. If there is no isolation and the atomicity property is implemented, the following isolation anomalies may occur [1 and 2].

- *The lost update anomaly* is by definition a situation where a first transaction reads a record for update without using locks. Subsequently, the record is updated by another transaction. Later, the update is overwritten by the first transaction. In extended transaction models, the lost update anomaly may be prevented, if the first transaction reads and updates the record in the same subtransaction using local ACID properties. Unfortunately, the read and the update are sometimes executed in different subtransactions belonging to the same parent transaction. In such a situation, a second transaction may update the record between the read and the update of the first transaction.
- *The dirty read anomaly* is by definition a situation where a first transaction updates a record without committing the update. Subsequently, a second transaction reads the record. Later, the first update is aborted (or committed), i.e. the second transaction may have read a non-existing version of the record. In extended transaction models, this may happen when the first transaction updates a record by using a compensatable subtransaction and later aborts the update by using a compensating subtransaction. If a second transaction reads the record before it has been compensated, the data read will be "dirty".
- *The non-repeatable read anomaly* or *fuzzy read* is by

definition a situation where a first transaction reads a record without using locks. Later, the record is updated and committed by a second transaction before the first transaction has been committed. In other words, it is not possible to rely on the data that have been read. In extended transaction models, this may happen when the first transaction reads a record that later is updated by a second transaction, which commits the update locally before the first transaction commits globally.

- *The phantom anomaly* is by definition a situation where a first transaction reads some records by using a search condition. Subsequently, a second transaction updates the database in such a way that the result of the search condition is changed. In other words, the first transaction cannot repeat the search without changing the result. Using a data warehouse may often solve the problems of this anomaly [3].

The countermeasure transaction model [12] describes countermeasures that reduce the problems of the anomalies. However, in section 4, *The Pessimistic View Countermeasure* is used to eliminate the risk of using dirty and/or the non-repeatable data, and therefore, the idea of this countermeasure is explained in the following. The pessimistic view countermeasure may be implemented by using:

- The pivot or compensatable subtransactions for updates that "limit" the users' options. That is, concurrent transactions cannot use resources that are reserved for the compensatable/pivot subtransaction.
- The pivot or retriable subtransactions for updates that "increase" the users' options. That is, concurrent transactions can only use increased resources after the increase has been committed.

For an example, when updating stocks, accounts, vacant passenger capacity, etc. it is possible to reduce the risk of reading non-available stock values. These pessimistic stock values will automatically be obtained if the transactions updating the stocks are designed in such a way that compensatable subtransactions (or the pivot transaction) are used to reduce the stocks and retriable subtransactions (or the pivot transaction) are used to increase the stocks. That is, for attributes with stock like values the compensatable subtransactions that reduce the stock may be used to "limit" the users' options and the retriable subtransactions that increase the stock may be used to "increase" the users' options.

2.4 The Durability Property

Updates of transactions are said to be *durable* if they are stored in a stable manner and secured by a log recovery system. In case a global transaction has the atomicity property (or relaxed atomicity), the global durability property (or relaxed durability property) will automatically be implemented, as it is ensured by the log-system of the local DBMS systems [12].

3. PROPERTIES OF HETEROGENOUS MODULAR ERP STRUCTURE

A traditional ERP system consists of modules like Sales, Production, Procurement, Accounting, CRM, HRM, etc. with the applications that are common for most types of industry. These modules may be extended with specific line of business modules like hospital sector, transport sector, and university administration in order to make it possible for the ERP system to function as an integrated system that support all the administrative functions of a specialized organization. In a traditional ERP system the whole database is common for all the modules of the ERP system even though special modules may have suppliers of their own. That is the applications of the ERP system may have full ACID properties. In a heterogeneous modular ERP architecture, each module may have its own database tables and applications. The applications of a module may access the tables of its module directly. In contrast, when an application in one module needs data from another module it must use e.g. SOA services and not access the data directly. Applications that use data owned by their module shall have full ACID properties. The applications of a module that use services from another module can only have relaxed ACID properties. Each ERP module has full autonomy over its own tables and its tables may be recovered independently from the tables owned by other ERP modules. The modules of a distributed modular ERP system may be mobile as it should be possible for them to operate in both connected or disconnected mode with respect to the other ERP modules depending on whether the ERP module has access to the tables of the other ERP modules or not.

It should be possible to have different versions of the same module. The different module versions may be developed by different software suppliers that have specialized in software for different lines of business. That is the modules may be heterogeneous. It should be possible to build ERP applications in the same way as package-deal houses by using different standard components/modules that may be delivered by different suppliers that in turn may use different databases.

In order to implement integration flexibility (fault tolerance in case a backup unit is used) between the ERP modules of different companies, it is not acceptable that the different modules communicate directly with each other. All communication between different modules must be executed by applications offered as e.g. SOA services. In order to implement relaxed ACID properties between the modules each module should offer the following types of services [8]:

- Read only services that are used when a local internet system wants to read data managed by another local internet system.
- Compensatable update services that are used when a local internet system wants to make compensatable updates in tables managed by another local internet system.
- Retriable update services that are used when a local internet system wants to make retriable updates in tables managed by another local internet system.

By using these three types of services from other local internet systems it should be possible for any local internet system to make distributed updates with relaxed ACID properties.

4. BENEFITS OF HETEROGENOUS INTEGRATED ERP MODULES

As described earlier the overriding benefit of using relaxed ACID properties across different application modules/databases is flexibility. Flexibility to migrate systems/modules without using a risky overnight conversion has been described by Frank, 2008. Flexibility to integrate mobile distributed users with stationary databases and the possibility for these users to operate in disconnected mode has been described by Frank, 2011d. The following examples of integrated heterogeneous ERP modules have previously been described in [5 and 8].

In a heterogeneous modular ERP architecture we define the Accounting module as the module that owns the Account, Account item, the Product stock tables and possible other tables where the balances have to be updated by compensatable and retriable subtransactions managed from many different modules. This is illustrated in the right part of the Figure 1.

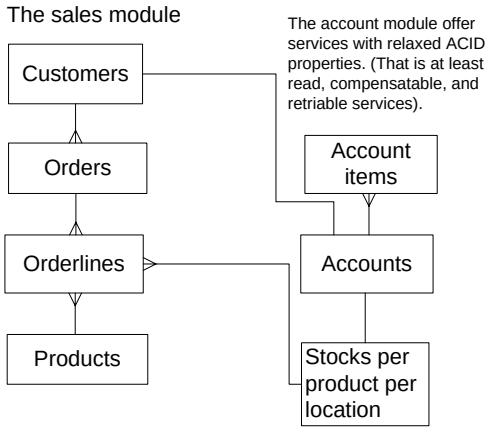

Figure 1. Illustration of the integration of the accounting tables with some sales module

A sales transaction may be implemented by first reading the customer record of the customer. Next, the customer/salesman may search the product table and for each selected product, compensatable subtransactions are used for creating an order-line, debit the customer account, reduce the product stock, and credit the product account for the products ordered. After having controlled possible additional payment conditions, the pivot subtransaction marks the order record as valid in the sales location.

If the order is cancelled later, a pivot subtransaction marks the order record as cancelled in the sales location, and initiates retriable subtransactions to credit the customer account, increase the product stock, and debit the product accounts for the products ordered. Account items must not be deleted, they may either be marked as cancelled or set off.

In this case, the Product stock and the Account tables are stored in the same database/module location, and therefore, the amounts in the Product stocks are consistent with the balances of the corresponding product accounts. In this example, it is assumed that the Pessimistic view countermeasure [12] is used to manage the dirty read anomaly that may occur when the Product amounts are read and updated. That is the Sales system, the Procurement system and the Production system cannot update the Product stocks directly as the applications of these systems do not belong to the accounting module with the Product stock table. However, the sales application shall use a compensatable SOA service to reduce the Product stocks, and the Procurement system can use a retriable SOA service to increase the Product stocks when new products arrive. In the same way, the Production system can use the retriable service to increase the Product stocks when new products have been produced and the compensatable service to reduce the stocks of raw materials when new productions are planned.

The Product stock table is also updated by the Procurement system and the Production system as these systems change the amount of products in the stocks. However, the integration of these modules with the accounting module may be implemented in the same way as the sales module.

The left part of Figure 2 illustrates how traditional E-government application modules may be integrated with the tables of an ERP accounting module [7].

The E- Government module may deal Tax cases where some of the Case items may involve tax payments. When a payment is received by the Tax system a compensatable subtransaction should debit the Account of the Citizen and a retriable subtransaction should credit the Account of the tax department. In a similar way, the Fine system of police authorities may receive fines from the convicted citizens. In Social cases, the Citizens may receive payments from social authorities. In this situation, a

compensatable subtransaction should first debit the Account of the social authorities. Next, a retriable subtransaction should credit the Account of the Citizen. In a similar way, other payments from E-government systems like Salary systems may take place.

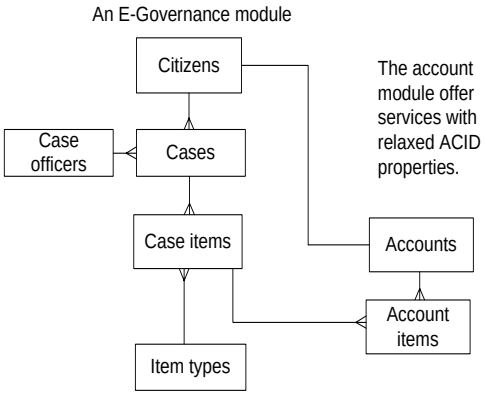

Figure 2. Illustration of the integration of the accounting tables with some E-Government module

5. DEVELOPMENT, IMPLEMENTATION AND OPERATIVE COSTS OF USING INTEGRATED HETEROGENOUS MODULES

In the previous sections an architecture for integrating heterogeneous ERP modules have been described. In this section, how this architecture may change the development, implementation, and operation of integrated modules/systems in order to get an overview of the changed costs is focused.

The development of application modules/ERP modules may take place in any location with access to the local database of the module. The only extra development costs are the costs of implementing the relaxed ACID properties across the boundaries of the heterogeneous modules. If the module is an ERP module these costs may be distributed on possibly many different ERP customers, and therefore, the costs may be more acceptable.

The implementation and test of a heterogeneous module may take place in any location with access to the local database of the module. The implementation and test of the integrated heterogeneous system may take place as if the integrated system is not distributed as the distribution is hided from the programmers. Therefore, it is possible to conclude that there are no extra costs in implementing and testing modules that use the integration architecture described in this paper.

Normally, one would believe that the operative costs of using our architecture are higher than those of traditional integrated homogeneous ERP modules. However, if SOA services are used locally they are not significantly more expensive than the use of local procedure calls that may be used when local ERP modules are integrated. On the other hand, if distributed SOA services are used they are more expensive than local procedure calls. However, we believe that ERP modules are distributed if the benefits of the distribution are larger than the costs. Therefore, it is possible to conclude that the operative costs of using our architecture are not significantly larger than using homogeneous integration architecture.

6. CONCLUSION AND FUTURE RESEARCH

In this paper, we have described an architecture for integrating heterogeneous modules from different suppliers by using relaxed ACID properties across the different modules. ERP systems are very complex software products. Therefore, we used ERP module integration as an example of how it is possible to integrate heterogeneous modules.

The overriding benefit of using relaxed ACID properties across different application modules/databases is flexibility to select the best of breed software for the different business functions of the company.

Even though the ERP modules may be distributed, the overriding extra costs of using our integration architecture are the development of relaxed ACID properties across the boundaries of the ERP modules.

We believe that in the future all users should have the possibility to be mobile and even have the possibility to operate in disconnected mode. In our view, the integration architecture described in this paper is well suited for such integration as retriable updates from remote mobile users may be asynchronous and have relaxed ACID properties.

REFERENCES

[1] Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E. and O'Neil, P. 1995. A Critique of ANSI SQL Isolation Levels. *Proceedings of ACM SIGMOD Conference*, 1-10.

[2] Breibart, Y., Garcia-Molina, H. and Silberschatz, A. 1992. *Overview of Multidatabase Transaction Management*. VLDB Journal, 2, 181-239.

[3] Frank, L. 2003. Data Cleaning for Datawarehouses Built on Top of Distributed Databases with Approximated ACID Properties. In *Proceedings of the International Conference on Computer Science and its Applications*. National University US Education Service, 160-164.

[4] Frank, L. 2004. Architecture for Integration of Distributed ERP Systems and E-commerce Systems. *Industrial Management and Data Systems (IMDS)*, 104(5), 418-429.

[5] Frank, L.2008. Smooth and Flexible ERP Migration between both Homogeneous and Heterogeneous ERP Systems/ERP Modules. *In Proceedings of the 2nd Workshop on 3rd Generation Enterprise Resource Planning Systems.* Copenhagen business School.

[6] Frank, L. 2010. *Design of Distributed Integrated Heterogeneous or Mobile Databases*. LAP LAMBERT

Academic Publishing AG & Co. KG, Germany, ISBN 978-3-8383-4426-3, 1-157.

[7] Frank, L. 2011. Architecture for ERP System Integration with Heterogeneous E-Government Modules, *Strategic Enterprise Resource Planning Models for E-Government: Applications & Methodologies.* Susheel Chhabra and Muneesh Kumar (Editors), IGI Global.

[8] Frank, L. 2011b. Architecture for Integrating Heterogeneous Distributed Databases Using Supplier Integrated E-Commerce Systems as an Example. In *Proceedings of the International Conference on Computer and Management (CAMAN 2011)*, Wuhan, China.

[9] Frank, L. 2011c. Countermeasures against Consistency Anomalies in Distributed Integrated Databases with Relaxed ACID Properties, In *Proceedings of Innovations in Information Technology (Innovations 2011)*. Abu Dhabi, UAE.

[10] Frank, L. 2011d. Architecture for Integrated Mobile Calendar Systems. *Mobile Computing Techniques in Emerging Market: Systems, Applications and Services*. A. V. Senthil Kumar, Hakikur Rahman (Editors), IGI Global.

[11] Frank, L. and Susanne Munck. 2008. An Overview of Architectures for Integrating Distributed Electronic Health Records. In *Proceedings of the 7th International Conference on Applications and Principles of Information Science (APIS2008)*. 97-300.

[12] Frank, L. and Zahle, T.1998. Semantic ACID Properties in Multidatabases Using Remote Procedure Calls and Update Propagations. *Software - Practice & Experience*, Vol.28, 77-98.

[13] Frank, L. and Louise Pape-Haugaard. 2011. Integration of Health Records by Using Relaxed ACID properties between Hospitals, Physicians and Mobile Units like Ambulances and Doctors. *International Journal of Handheld Computing Research (IJHCR)*, IGI Global.

[14] Garcia-Molina, H. and Salem, K. 1987. *Sagas.* ACM SIGMOD Conference., 249-259.

[15] Garcia-Molina, H. and Polyzois, C. 1990. Issues in disaster recovery, *IEEE Compcon.*, IEEE, New York, 573-577.

[16] Gold-Bernstein, B. and William A. Ruh. 2005. *Enterprise integration: the essential guide to integration solutions*. Addison-Wesley.

[17] Hohpe, G. and Bobby Woolf. 2004. *Enterprise Integration Patterns*. Addison-Wesley.

[18] Linthicum, D. 2004. *Next generation application integration.* Addison-Wesley.

[19] Mehrotra, S., Rastogi, R., Korth, H., and Silberschatz, A, 1992. A transaction model for multi-database systems. In *Proceedings of International Conference on Distributed Computing Systems*. pp. 56-63.

[20] Zhang, A., Nodine, M., Bhargava, B. and Bukhres, O. 1994. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems . In *Proceedings of ACM SIGMOD Conference*. pp. 67-78.