28.8.2008

Smooth and Flexible ERP Migration between both Homogeneous and Heterogeneous ERP Systems/ERP Modules

Lars Frank
Department of Informatics, Copenhagen Business School, Howitzvej 60, DK-2000 Frederiksberg, Denmark. Phone: +45 38 15 2400, Fax: +45 38152401, e-mail: lpf.inf@cbs.dk

Abstract

It is very risky to migrate/convert from one ERP (Enterprise Resource Planning) system to another because normally companies can only survive a short time without a running ERP or legacy system. In this paper, we will describe how the architecture of a distributed modular ERP system can be used to migrate from one ERP system to another over a longer or shorter time period.
A *distributed modular ERP system* is a 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. All data exchange between modules must be executed by using applications like SOA services. In a distributed modular ERP system, 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.
In this paper, we will describe how it is possible by using the architecture of a distributed modular ERP system to migrate from a traditional ERP system to a distributed modular ERP system by migrating the modules one by one.
In a distributed modular ERP system, it is easier to integrate special line of business ERP modules as the modules may be heterogeneous and have different suppliers.
It is also easier to prevent bottlenecks in a distributed modular ERP system than in a central ERP system as any of the modules may be upgraded independently of the others. Therefore, it may be convenient to keep the distributed modular ERP architecture even after the migration period is over.

Keywords: ERP migration, ERP flexibility, risks management, distributed modular ERP system.

1. Introduction

In a *distributed ERP* (Enterprise Resource Planning) *system*, the different local ERP systems are integrated in such a way that each local system can use the resources/stocks managed by the other local ERP systems in any location (e.g. Frank, 2004). That is, from an ERP location where a product is out of stock, it is e.g. possible to use the stock in another ERP location. While the architecture of distributed ERP systems are used for integrating different ERP locations, e-commerce systems, and mobile ERP systems (e.g. Frank, 2008) the architecture of a distributed modular ERP system is used for migrating from one ERP system to another over time. That is a risky over night migration may be avoided.
The transactions of a distributed modular ERP system do not have the traditional ACID properties (Atomicity, Consistency, Isolation, and Durability) properties. Such transactions have only relaxed ACID properties (Frank, 2008). In the extended transaction model used in this paper the relaxed

ACID properties are implemented in the following way: The global atomicity property is implemented by using retriable, pivot and compensatable subtransactions in that order. The global consistency and isolation properties are managed by using countermeasures (Frank and Zahle, 1998), and the global durability property is implemented by using the durability property of the local DBMS (Data Base Management System) systems.
Applications with relaxed ACID properties function from a user point of view as if the traditional ACID properties were implemented by the underlying DBMS systems. However, in reality the relaxed ACID properties are implemented as an integrated part of the applications. Therefore, it does not matter whether all the different ERP modules use the same DBMS system or not.

In order to avoid the problem that old applications from the traditional ERP system cannot run because some of the involved tables have been migrated it is important that tables with stocks of money (account tables) or products (stock tables) that are used by many different modules are migrated last and therefore, it is necessary to make (SOA) services for these tables that make it possible to update the balances of these tables by using retriable and/or compensatable subtransactions. For the new migrated modules it does not matter much where data are stored as long as it can be accessed by e.g. SOA services, and therefore, the new distributed modular ERP applications can use the stock and accounting tables of the old ERP system. Other tables like the employee table and the organization/department table that may be used by applications from many different modules are updated relatively rare, and therefore, these tables may be updated manually (or replicated automatically) in both ERP systems in the period of migration.

The paper is organized as follows: First, we will describe the transaction model used in this paper. Next, we will describe the properties of a distributed modular ERP system that are necessary for migrating ERP modules independently of each other. Next, we will use the sales and accounting modules as examples to illustrate how it is possible to migrate to another central or distributed ERP system by using a distributed modular ERP system. In section 5, we will describe how an ERP supplier can develop and use a distributed modular ERP system to conquer new marked shares by making services to the old ERP system that support relaxed ACID properties to the new ERP system. In section 6, we will illustrate by examples how specific line of business modules may be integrated in a distributed modular ERP architecture by only exchanging the modules that are line of business specific. Finally, we have conclusions and future work.

Related Research: The transaction model described in section 2 is *the countermeasure transaction model* (Frank and Zahle, 1998). This model owes many of its properties to e.g. Garcia-Molina and Salem, 1987; Mehrotra et al., 1992; Weikum and Schek, 1992 and Zhang, 1994. Like Zhang et al. 1994, we might just as well have called our transactions '*flexible transactions'* as the only new facility is the use of countermeasures to manage the missing consistency and isolation properties. However, as it is our experience that finding countermeasures against isolation anomalies across different ERP databases is not a major problem, we will not deal with the problem in this paper. An architecture for distributed ERP systems is described by e.g. Frank, 2004, and the current paper is a specialization of this architecture where the integration of ERP modules are taken into account.

2. The Transaction Model

A *multidatabase* is a union of local autonomous databases. *Global transactions* (Gray and Reuter, 1993) 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 (Frank and Zahle, 1998) 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.

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 (Berenson et al., 1995 and Breibart, 1992).

- *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 (Frank, 2003).

The countermeasure transaction model (Frank and Zahle, 1998) describes countermeasures that reduce the problems of the anomalies.

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 (Breibart et al., 1992).

3. Description of the Properties of a Distributed Modular ERP system

A traditional ERP system consists of modules like Sale, 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 e.g. the hospital sector, the 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 distributed modular ERP system, each module may own 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 only use data owned by their module may have full ACID properties. The applications of a module that use services from another module can only have relaxed ACID properties because the services of a module must only use short duration locks, where no data is locked across the boundaries of the module. Therefore, it is only possible to have relaxed ACID properties between the different possible heterogeneous databases that are used by the different modules.

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.

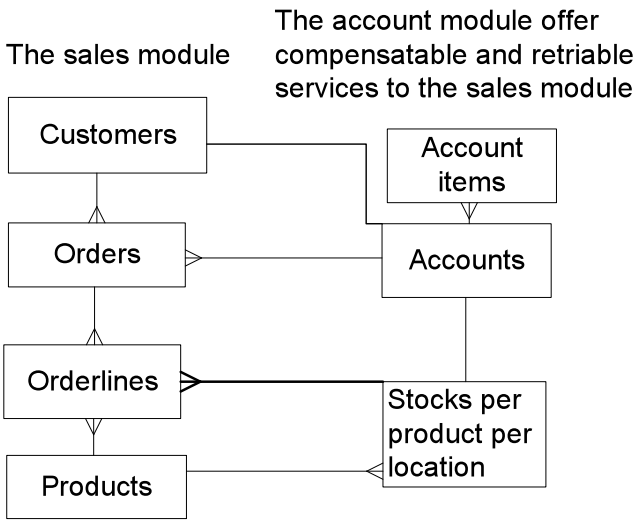
It should be possible to have different versions of the same module. The different module versions may be developed by different software suppliers that have specialized in software for different types of industry. That is the modules of a distributed modular ERP system may be heterogeneous. It should 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 the properties described above, the following rules must be followed.

- Read requests from other modules must only use short duration locks, and therefore, countermeasures against the isolation anomalies should normally be used in such situations.
- In order to make the modules as autonomous as possible, the pivot subtransactions of the applications of a module should always be executed in the location of the module. That is, all modules must only offer read only, compensatable, and retriable services to be used from other modules.

4. Description of How to Migrate by Using a Distributed Modular ERP system

In a distributed modular ERP system 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. In order to prevent having different versions of the balances it is important that the tables and applications of the accounting module are the last to be migrated from the old accounting system. To make it possible for the old accounting module to function through the whole period of migration it may be necessary to have replicated versions of e.g. tables with Organization/department information as these tables may be necessary in both the new distributed modular ERP system and in the old ERP accounting system.
The balance of the Accounting table may be updated by applications from almost all the other modules, and the Product stock table is at least updated by the Sales system, the Procurement system, and the Production system as these systems change the amount of products in the stocks. Therefore, the Accounting and the Product stock tables must have e.g. a compensating SOA service to reduce the balance/amount attributes of these tables from the location of another module. Likewise, it is necessary to have e.g. a retriable SOA service to increase the balance/amount attributes of the tables from the location of another module.
As soon as the compensatable and retriable service interfaces to the old ERP system are made, it is possible to migrate the Sales system, the Procurement system, and the Production system in any order.
The following ER diagram illustrates the integration of the tables of the new sales module with the tables of the old ERP system.



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 later is cancelled, 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. As account items must never 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, the Pessimistic view countermeasure (Frank and Zahle, 1998) has been used to manage the dirty read anomaly that may occur when the Product amounts are read. This is also normal practice when stocks are limited and the transactions are long lived. For example, airline tickets may be reserved, changed, and re-changed over a longer period of time before the final purchase is committed. Many different airline companies with different heterogeneous databases may be involved, and therefore, there is no tradition for having full ACID properties for such transactions.
In this case, 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 can e.g. use a compensatable SOA service to reduce the Product stocks, and the Procurement system can a use 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.
As described earlier, we will recommend that the accounting module is migrated last to prevent having more than one version of stock amounts and account balances. Tables like Employee and organization/department tables where updates are relatively rare may be created in both the new and the old system because it does not matter if updates are delayed a little and such updates may therefore be replicated without much extra work. The order of ERP module migration should be determined in such a way that the applications of migrated modules only use migrated tables, replicated tables, or non-migrated tables with retriable and compensable update services. In the same way, the non migrated applications must only use non migrated or replicated tables in the period of migration. Therefore, e.g. the CRM (Customer Relations Management) module should be migrated after the Sales system as the Customers and Orders will be used by the CRM module. Likewise, just before the accounting module is migrated, the old accounting applications can only use the accounting tables and the replicated tables. However, the new accounting module may use any tables because just before the new accounting module is put into operation all the rest of the ERP tables must be converted to the table format of the new ERP system.

5. How can an ERP Supplier Develop and Use a Distributed Modular ERP System

A distributed modular ERP system has not been developed yet but it should be easy and flexible for an ERP supplier to start the development of a distributed modular ERP system. The reason why is that it is easy for an ERP supplier to make the compensatable and retriable service interfaces to the old ERP system, and then make and test the modules of the new distributed modular ERP system, module by module in the order they prefer. Actually, it is only the integration architecture the ERP supplier needs to change, and therefore, the implementation may be relatively fast if no new functionality is implemented. The reason for such an implementation strategy may be the benefits of more flexibility. First, it may be easier to implement specific Line of Business ERP Modules as described in the next subsection. Next, it may be easier to develop new functionality in the ERP modules as the modules are more independent of each other, and therefore, the complexity of the total ERP system may be reduced. Finally, it may be possible to increase the capacity of the total ERP system as the capacity of the modules may be independent of each other. Using the architecture of a distributed modular ERP system will increase the load on the CPUs and the disks

but the load may be distributed on many smaller CPUs and their connected discs, and this may be cheaper than running all applications on a central computer. Anyway, it is easier to prevent bottlenecks in a distributed system than in a central system as any of the subsystems may be upgraded independently of the others. Therefore, it may be convenient to keep the distributed modular ERP architecture even after the migration period is over.

6. Integration of Specific Line of Business ERP Modules

Almost all organizations need ERP modules for Accounting, Procurement, HRM (Human Resource Management), and CRM (Customer Relations Management) where the "customers" may not be paying customers but anyone who have contact with the organization coursed by the mission of the organization. The differences between types of industry are often mirrored in the production module, especially if the "customer" forms part of the production. This is for example the case in hospitals where products like diagnoses, surgeries, medication, etc, cannot be produced without the "customer". A lot of research has been done in describing how EHR (Electronic Health Records) may be integrated across different health databases (e.g. Frank and Munck, 2008).
Another example is education systems, where the "customers" are students and the products are courses that may be consumed by the "customers" with more or less qualifying results.
In both these cases, the "Order lines" are customer treatments/actions that may be executed in different locations like hospital or universities. We believe that the architecture of a distributed modular ERP system may benefit the integration of such systems in the following areas:

- The architecture supports access to the "Order lines" created in remote locations.
- The architecture supports asynchronous replication of "Order lines" created in remote locations as described in (Frank and Munck, 2008).
- The architecture supports integration of heterogeneous modules even though the "Order lines" of such systems may be more or less incompatible with each other (Frank and Andersen, 2008). The development of standards may help in reducing the problem of integrating incompatible data. However, as long as there is more than one standard it is important to have solutions that can deal with the problem.

7. Conclusions and Future Research

In this paper, we have described the architecture of a distributed modular ERP system. By using a distributed modular ERP system it is possible to have more flexibility in migrating from one ERP system to another. As each ERP module may have its own database it is also possible to increase the capacity of the total ERP system by avoiding DBMS bottlenecks. It is also easier to integrate special line of business ERP modules as the modules may be heterogeneous and have different suppliers.
ERP systems are very complex, and we have not described all the details of making an ERP migration. Therefore, we will in our coming research go into more details about the needs each type of ERP module has to the migration process.
Another future research objective is to analyze what specific line of business distributed ERP modules may have in common and what special needs specific line of business modules may have to the common ERP modules that use the distributed modular ERP architecture.

References

Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E. and O'Neil, P., 1995, "A Critique of ANSI SQL Isolation Levels", *Proc ACM SIGMOD Conf.*, pp. 1-10.

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

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

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

Frank, L, 2007, 'Smooth ERP Migration by Using Next Generation Distributed ERP Systems', Proc of the IRMA (Information Resources Management Association) International Conference, Vancouver, pp 1026-1027.

Frank, L. and Susanne Munck, 2008, 'An Overview of Architectures for Integrating Distributed Electronic Health Records', Proceeding of the 7th International Conference on Applications and Principles of Information Science (APIS2008), pp297-300.

Frank, L. and Stig Kjær Andersen, 2008, 'Integration of Operative Distributed Electronic Health Records by using Datawarehouse Design Techniques', Working paper, Copenhagen Business School.

Frank, L. and Zahle, T, 1998, "Semantic ACID Properties in Multidatabases Using Remote Procedure Calls and Update Propagations", *Software - Practice & Experience*, Vol.28, pp77-98.

Garcia-Molina, H. and Salem, K., 1987, "Sagas", *ACM SIGMOD Conf*, pp 249-259.

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

Gray, J. and Reuter, A., 1993, "Transaction Processing", *Morgan Kaufman*, 1993.

Mehrotra, S., Rastogi, R., Korth, H., and Silberschatz, A., 1992, "A transaction model for multi-database systems"*, Proc International Conference on Distributed Computing Systems*, pp 56-63.

Polyzois, C. and Garcia-Molina, H., 1994, "Evaluation of Remote Backup Algorithms for Transaction-Processing Systems", *ACM TODS*, 19(3), pp 423-449.

Weikum, G. and Schek, H., 1992, "Concepts and Applications of Multilevel Transactions and Open Nested Transactions", *A. Elmagarmid (ed.): Database Transaction Models for Advanced Applications*, Morgan Kaufmann, pp 515-553.

Zhang, A., Nodine, M., Bhargava, B. and Bukhres, O., 1994, "Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems", *Proc ACM SIGMOD Conf,* pp 67-78.