1993

Supporting Distributed Transaction Dependencies and Security Constraints: A knowledge Base Approach

Noureddine Boudriga

Omran Bukhres

Report Number:

93-021

SUPPORTING DISTRIBUTED TRANSACTION DEPENDENCIES AND SECURITY CONSTRAINTS: A KNOWLEDGE BASE APPROACH

Noureddine Boudriga
Omran Bukhres

CSD-TR-93-021
March 1993

Supporting Distributed Transaction dependencies and Security Constraints: A Knowledge Base Approach

Noureddine Boudriga[1], Omran Bukhres[2]
1. Math Dept., Faculty of Science, Tunis, 1060 Tunisia
nab@spiky.rsinet.tn
2. CS Dept., Purdue University, West Lafayette, IN 47907
bukhres@cs.purdue.edu

Abstract

For many applications, a distributed system is an attractive alternative to a single system because it supports global applications accessing multiple systems, and thus enhances performance. The rapid growth of advanced applications involving distributed transaction processing has resulted in the development of various distributed systems, models and transaction languages. In this paper, we present the InterBase system, its Parallel language (IPL), and its Interbase Logic Controller (ILC). IPL is a transaction-oriented language that allows users to write global transactions by specifying all associated actions and their sequences, as well as logical dependencies and data flows among subtransactions. ILC is used to control the different tasks to be performed within the Interbase environment, all without violating the autonomies of the local systems and respecting their heterogeneities. ILC guarantees two levels of consistency, consistency among transaction dependencies and consistency among security policies.

Keywords. Distributed database, Knowledge Database, Multilevel Security, Interdependent data, Flex transaction.

1 Introduction

The rapid growth of advanced applications involving distributed transaction processing has resulted in the development of many distributed models and languages. Early distributed systems were programmed in conventional sequential languages, were centralized, and support limited transactions. The traditional programming languages have serious limitations and constraints in the representation and the execution of distributed applications. With today's development of sophisticated and complex distributed applications, the extended traditional sequential languages cannot adequately support the development of these advanced applications. Moreover, properties such as isolation and consistency have to be relaxed in the new distributed system in order to satisfy other needs like autonomy and heterogeneity.

One of the consequences of the information explosion taking place in society is the emerging need to access heterogeneous and isolated repositories. Due to the isolation property, it is more difficult for programmers to write global applications which make full use of the data and resources at their disposal since the systems that they need to use are not integrated.

The transaction model defined within the Interbase (a multidatabase system project at Purdue UNiversity) environment, the Flex model, provides for additional capabilities not originally forseen in traditional transactions. These new capabilities are required in order to describe applications in multidatabase systems. Among these capabilities, we mention:

Functional replication. Alternative ways by which a specific task can be performed are conveniently stated in the in the Flex model.

Control Isolation. The Flex transaction model allows transactions to include some transactions that are compensatable.

Dependency. The model allows for specifying functions and relations that can be used to influence the execution of a transaction.

The Interbase Parallel Language fully supports a distributed programming environment. It supports a high degree of parallelism and provides synchronization and high-level communication among subtransactions within a global transaction.

Transaction management in multidatabase systems has been the subject of extensive research. Many problems remain unresolved because of the complexity caused by data distribution, heterogeneity, the need to preserve the autonomy of the member database systems, and the security policies of each local database system.

Interdependent data are data related to each other through integrity constraints. Interdependency has been found to occur naturally in organizations, and is costly to maintain (see, for example, [3]). Examples of interdependent data include replicated data, partially replicated data, and summary data. Various classifications and issues related to interdependent data management have been done, based on several criteria such as type of interdatabase dependency, degree of local autonomy, and data consistency criteria.

A multilevel secure database system is a system that protects data classified at more than one security class and allow sharing between users with different clearance levels. Permission to access data is determined not only by the accessibility of the user requesting access to the data, but also by the security level of the data. The clearance level of the user classification can be applied at different levels of granularity in the database, for example, at the relation, tuple, attribute, or element level [1]. The interdependence of data and potential difference between the security policies adopted by the system members make it difficult to achieve a secure system.

In this paper, we describe the role of logic to perform two tasks: controlling dependency between subtransactions and resolving security inconsistencies within interdependent data. These two tasks are performed through the components of the InterBase system. The InterBase Parallel Language (IPL), supports a powerful description of advanced transactions, provides communication among the subtransactions within a global transaction, and allows for properties such as compensatability and function replication. The Distributed Interbase Transaction manager interprets and coordinates the execution of global transactions. The rest of the paper is organized as follows: A description of advanced features of the transactions and a presentation of the Interbase Parallel Language is provided in section 2. Section 3 presents the The multilevel security of interbase and discusses the inconsistency of security constraints. Section 4 outlines the structure of a Knowledge Database, and describes the interbase system. The concluding remarks and an agenda for future work appear in Section 5.

2 InterBase Parallel Language

IPL supports distributed applications. It allows the parallel execution of Flex transactions. We present in this section transactions dependencies, the definition of Flex transaction, the structure of IPL programs, and the notion of acceptable sets.

2.1 Transactions Dependencies

An object in a database has a type, a state, and a set of operations that provide the means to create, modify and retrieve the state of the object. The state of an object is represented by its content. A global transaction accesses and manipulates the objects in a local database by submitting a subtransaction that invokes operations specific to the objects. The effect of an operation invoked by a subtransaction on an object is made permanent if the subtransaction is commited; it is deleted if the subtransaction is aborted.

Dependency relations provide a convenient way to describe the behavior of subtransactions and can be expressed in terms of subtransaction states. The state of a subtransaction is time dependent and has six values: waiting state, executing state, success state, failure state, committed state, and aborted state. Different types of dependency can occur, among them we mention [6], [7]:

Success Dependency. Transaction t is success dependent (t $<_{SD}$ t') if t can be executed only after t' is successfully executed

Failure Dependency. Transaction t is failure dependent(t $<_{FD}$ t') if t can be executed only after t' is executed and failed

Commit Dependency. Transaction t is commit dependent on t' (t $<_{CD}$ t') if t and t' commit then the commitment of t' precedes the commitment of t.

Abort Dependency. Transaction t is abort dependent on t' (t $<_{AD}$ t') if when t' aborts, then t aborts

Exclusion dependency. Transactions t and t' are exclusive dependent (t ED t') if both t and t' cannot commit.

The above dependencies are classified behavioral dependencies. Such dependencies describe relationships among (sub)transactions based on their behavior (i.e. the different states of the subtransactions). Two other categories can be established, structural dependencies and external dependencies.

A structural dependency describe the hierarchy among the subtransactions of a global transaction. In general this hierarchy is a tree-like structure with the global transaction as the root. While traditional transactions are represented by a one level tree, Flex transactions are represented by a two level tree, and a nested transaction has no height limitation.

An external dependency associates the different states of a (sub)transaction to external parameters such as time, cost values, etc.. Function passing between substransactions can be considered a behavioral dependency.

Illustrative Example 1. Consider a travel agent (TA) information system [8]; a transaction t in this system may consist of the following tasks:

c_1: TA negotiates with airlines for flight tickets, and get a ticket if the price. is less than $300.

c_2: TA negotiates with car rental companies for car reservations, and reserve a car if flight ticket is purchased.

c_3: TA negotiates with hotels to reserve rooms.

Let us assume that the refinment of these subtransactions leads to the following subtransactions:

t_1: Order a ticket at Northwest Airlines;

t_2: Order a ticket at United Airlines;

t_3: Rent a car at Hertz

t_4: Rent a car at Avis;

t_5: Reserve a room at Sheraton;

t_6: Reserve a room at Ramada.

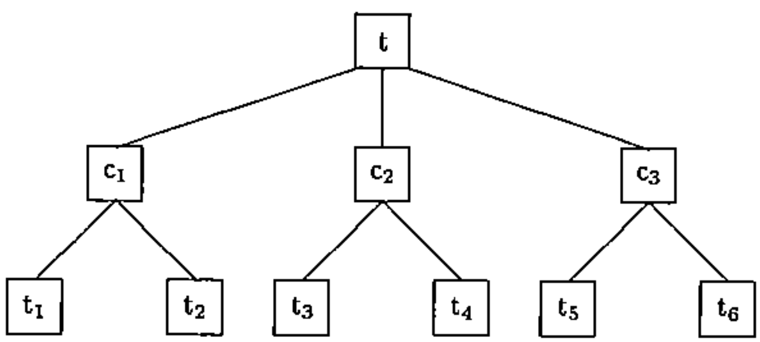

Figure 1. TA Transaction

The different dependencies are described by:

$<_{SD}$ and $<_{FD}$: $t_i <_{SD} t_k$ $t_j <_{FD} t_1$ for j=3,4 i=3,4 and k=5,6

ExtD: cost(t_i) < $300, for i=1,2.

2.2 Flex Transactions

The Flex transaction model is designed to provide more flexibility in transaction processing. It allows the description of transaction that is composed of a set of task. Each task is achieved through a set of functionally equivalent subtransactions. The execution of a Flex transaction succeeds if all its tasks are accomplished. A Flex transaction is resilient to failures in the sense that it may proceed and commit even if some of its subtransactions fail. The Flex transaction model allows the specification of dependencies on the subtransactions. The most useful dependencies in the Flex transaction model are *failure-dependencies, success-dependencies* and *external dependencies.*

In order to capture the notion of the compensability of subtransactions, we use the concept of type; a subtransaction is said to be of type C if it is compensatable; it is of type NC if it is non-compensatable. Call T the set of subtransactions of a transaction denoted by t. and t is called a Flex transaction if there is a 3-tuple (DEP, ExD, Acc) such that: DEP is a set of internal dependencies defined on T; ExD is a set of external dependencies defined on the elements of T; and Acc is a boolean function, called the acceptable function, which defines the different combinations of subtransaction states that are acceptable to commit transaction t.

We illustrate the definition of Flex transaction by using the example of the travel agent transaction introduced in Figure 1. In that case, we have T = $\{t, c_1, c_2, c_3, t_1, t_2, t_3, t_4, t_5, t_6\}$; DEP is composed by the two dependencies $<_{SD}$ and $<_{FD}$ defined above; Ext is reduced to one relation defined by Cost(t_i) < \$300, for i=1,2; and Acc is the function that gives the value "acceptable" to the sets $\{t_1, t_3, t_5\}$ and $\{t_2, t_3, t_6\}$.

In the sequel we represent an acceptability function by the enumeration of all the sets to which it associates the value "acceptable".

2.3 Structure of IPL programs

An IPL program describes a Flex transaction. It contains four fundamental components: objects and types, subtransaction definitions, dependency descriptions among subtransactions, and acceptable function.

Objects and types. Objects in IPL serve as results of and arguments to subtransactions in the context of a global transaction. Therefore, in IPL each subtransaction is associated with a type. Types have unique names and are used to categorize objects into sets capable of participating in a specific set of subtransactions.

Definition of Subtransactions. A subtransaction is a task executable by a local software system in a distributed system. The subtransaction may require the results of other subtransactions as its input. It may also be executed under particular time constraints or other conditions. A subtransaction is provided with an identifying name which should be unique within the context of a global transaction.

Dependency Description. Dependency description provides users with a mechanism for specifying the explicit dependencies among the subtransactions of a global transaction. That is, the execution order of the subtransactions of a global transaction can be defined with the use of the IPL dependency description. Correct parallel execution and synchronization among the subtransactions of a global transaction can thus be specified through the dependency description. For example, given six subtransactions t_1, t_2, t_3, t_4, t_5, and t_6, their execution order, and IPL dependency description are defined by:

1. t_2 will be executed only if t_1 succeeds.
2. t_3 will be executed only if t_1 fails.
3. t_4 and t_5 will be executed only if t_2 or t_3 succeeds.
4. t_6 will be executed only if t_3 and t_4 succeed or t_5 fails.
5. the global transaction will succeed if at least two of t_4, t_5, and t_6 succeed.

Acceptable sets. The fourth component of IPL begins with the keyword **acceptable_sets** and ends with the keyword **endaccs**. The acceptable sets provide function replication which can tolerate the failure of individual subtransactions by exploiting the fact that a given function can frequently be accomplished by more than one software system. For example, the transaction programmer may leave to the system the choice of renting a car from Hertz or Avis.

An acceptable set consists of a subtransaction list and a sufficient acceptable condition of the global transaction. When a global transaction reaches its final status, the user is asked to select a preferred acceptable set from an array of alternatives. All the subtransactions in an acceptable set in the array must be successful. Successful non-compensatable subtransactions are maintained in an uncommitted state until the global transaction is completed. When the user chooses an acceptable set and the global transaction commits, the uncommitted subtransactions in the acceptable set then perform their commit operations, all other uncommitted subtransactions perform their abort operations, and the compensatable subtransactions not in the acceptable set perform their compensating operations. When the global transaction decides to abort, all the successful subtransactions perform their abort or compensating operations.

Acceptable sets support function replication within global transactions, and thus enable them to tolerate the failure of individual subtransactions by exploiting the ability of several software systems to accomplish a given function. For the example presented in section 2.1, the acceptable sets could be:

acceptable_sets
$\{t_1, t_2, t_4, t_5\}$, $\{t_1, t_2, t_4, t_6\}$, $\{t_3, t_4, t_5, t_6\}$, $\{t_3, t_4, t_5\}$, $\{t_3, t_4, t_6\}$
endaccs

In this example, five acceptable sets are included; they are subtransaction sets $\{t_1, t_2, t_4, t_5\}$, $\{t_1, t_2, t_4, t_6\}$, $\{t_3, t_4, t_5, t_6\}$, $\{t_3, t_4, t_5\}$, and $\{t_3, t_4, t_6\}$.

The success of any of these five subtransaction sets will result in the success of the global transaction, and thus provide function replication within the global transaction.

2.4 The InterBase Logic Controller

In this section we describe the ILC component for IPL. The function of this component is to insure the consistency of transactions dependencies and to build acceptable sets for transaction.

Dependencies. The different dependencies are not disjoint and may have some overlaping semantics. The Interbase approach to resolve the potential inconsistencies between these dependencies is to establish a knowledge base that reports on the different dependencies stored. Among the rules stored, we mention:

Transitive Rule ($<_{SD}$ is an example of such transitive rule)

If R is transitive **and** t R t' & t' R t", **Then** t R t"

Symmetric Rule (ED is an example of such transitive rule)

If R is symmetric **and** t R t', **Then** t' R t

Overlapping Rule

If R includes R' **and** t R t', **Then** t R t'

Disjoint Rule ($<_{SD}$ and $<_{SD}$ represent an example of such rule)

If R is disjoint from R' **and** t R t', **Then** not(t R' t')

These rules complete the definition of the different dependencies that a user can utilize and check for the inconsistency between them.

Acceptable Sets. With IPL programming the user is allowed to express the acceptable sets. However these acceptable sets should be consistent with the success dependency and failure dependency relations. Theoretically the acceptability function is deduced from these relations using a set of rules among which we mention the following:

Terminal_Acc rule

If S acceptable set **and** $\{t \mid t <_{SD} x\} = \emptyset$ **and** $\{t \mid t <_{FD} x\} = \emptyset$, **Then** x $\in$ S

Success_depend rule

If S acceptable set **and** x$\in$ S **and** y$<_{SD}$ x , **Then** y$\in$ S

Failure_depend rule

If S acceptable set **and** x$\in$ S **and** x$<_{SD}$ y, **Then** y$\in$ S

The list of rules in that paper is not exhaustive. Other dependencies may have effect on the determination of the acceptable sets. Applying these rules to the TA transaction example, we can see that, for the TA transaction, only 8 acceptable sets can be defined:

$$\{t_i, t_j, t_k\}, \text{ for i=1,2, j=3,4, k=5,6}$$

3 On the Security of InterBase

A multilevel database system is a database system that protects data classified at more than one security class and allow sharing between users with different clearance levels of the user. Permission to access a data is determined by not only the accessibility of the user requesting access to the data, but also the security level of the data. The clearance level of the user classification can be applied at different levels of granularity in the database, for example, at the relation, tuple, attribute, or element level [1]. Security level of each data may be assigned explicitly, by attaching a label of security to the data, or implicitly, by defining a set of security constraints. An effective security policies for a multilevel distributed database system should ensure that users only acquire the information to which they are authorized. The Bell-LaPadula security model [2] is used as such a security policy.

Security Constraints are the rules which assign classification levels to data. They consist of data specification and a classification. The data specification defines any subset of the database; the classification defines the security level of each element of this subset. We address two types of classifications:

1. **Simple classifications:** They assign security levels by tuple and by element as they are stored in the database.
2. **Context classifications:** They assign security levels to the result of applying functions on an attribute or subset of attributes, or change the security levels of the data upon changing factors such as time.

We consider the set SEC of all security constraints. An element of SEC is a pair (spec, class), composed of data specification "spec" and the classification "class" of this data. Since a set of global security constraints must be based on consistent local security constraints, our system must detect and resolve all inconsistent security constraints. Among the specifications inconsistencies specifications, we find:

1. **Conflicting Security Constraints.** Conflicting security constraints are those constraints that classify the same fact into different classifications.
2. **Included Security Constraints.** Included security constraints are those constraints that are enforced to the relations or attributes that are the same or structurally equivalent.

3. **Disjointed Security Constraints.** Disjointed security constraints are those constraints that are enforced to some related but not the same relations or attributes.
4. **Dominated authority levels.** A classification C of data specification A dominates C' if C is higher than C'.

A longer list of inconsistencies can be established. Based on this list, we define a rule base that describes how to correct the incriminated inconsistencies. We discuss in the following the rules that solve the four inconsistencies mentioned above.

Illustrative Example 2. Consider an HDDBS consisting of two LDBSs, D_1 and D_2 . Let R_1 and R_2 be two relations at D_1 and D_2, respectively:

R_1(ProjNo, ProjName, Budget) and R_2(P_No, P_Name, P_Researchers)

We consider the following security constaints enforced on R_1 and R_2, respectively:

C_1: The project name is confidential (at D_1);

C_2: The project name is secret (at D_2);

C_3: The project name is top-secret (at D_1) if its budget is greater than 1 million dollar;

C_4: The project name is top-secret (at D_2) if it involves more than ten researchers;

C_5: The project name is top-secret (at D_2) if its budget is greater than 2 million dollar;

Then C_1 and C_2 conflict, C_3 and C_4 are disjointed, and C_3 includes C_5.

The ILC component for Security Inconsistency. The function of this component is to locate and resolve inconsistencies between security constraints. The approach is based on the use of a database rule. In the following, we present some of these rules. Let S be the set of ssume security constraints, and assume that (A,C) and (A',C') are in S.

Conflicting_Security Rule

If A and A' are interdependent **and** C$\neq$C' , **Then**

Included_Security Rule

If A and A' are included (or overlapped), **Then** change (A',C') to (A,C')

Dominated_authority Rule

If if A=A' and C$\geq$C', **Then** remove (A',C') and change C to include C'

4 The InterBase System

In this section, we present the ILC, and briefly describe the InterBase System, and then discuss the key components of the Distributed InterBase Transaction Manager. The correct interaction among concurrent global transaction and recovery issues are also discussed.

4.1 The Interbase Knowledge Base

The Interbase Knowledge has three components: 1) A rule base that describes the relationship between the dependencies available for the user. 2) A rule base that contains rules to check the reachability of acceptable sets and the automated generation of all acceptable sets. 3) A rule base that controls the consistency of security constraints. All the rules represented in the Interbase Knowledge base are production rule-like, and the system works in forward reasoning.

While the rule base for acceptable sets is limited in size, the rule base for security has no limitation, because of the interdependence between data. Nevertheless, The complexity of the rules represented in ILC dependes on the nature of interdependency of the data.

4.2 Architecture of InterBase System

We will describe the various components and modules of the system and briefly explain their mutual interactions. The InterBase System is designed to allow users to write global applications over a distributed, autonomous, and heterogeneous computing environment (in particular, a multidatabase environment), while retaining the autonomy of Local Software Systems (LSSs).

The major components and modules of the InterBase System and the relationships among them are presented in Figure 2. At present, the InterBase System runs on an interconnected network with a variety of hosts that include Sun, HP and NeXT workstations, Sequent machines, IBM mainframes, and IBM/PCs.

The Distributed InterBase Transaction Manager (DITM) is at the center of the InterBase System. DITM interprets and coordinates the execution of global transactions, which are in IPL format, over the entire system.

RSIs ensure a uniform interface to DITM and deal with the heterogeneity of the LSSs, thus relieving DITM from dealing with each LSS directly.

An IPL text from either source is executed by DITM as a global transaction over the InterBase System. Assisting in this process is the Decentralized Concurrency Controller (DCC), consisting of a Group Manager (GrMn) and Subtransaction Schedulers, each of which is a portion of an RSI. DCC is so named because it is based on decentralized algorithms discussed in [4]. DCC is used to manage the parallel access of global transactions over the InterBase System.

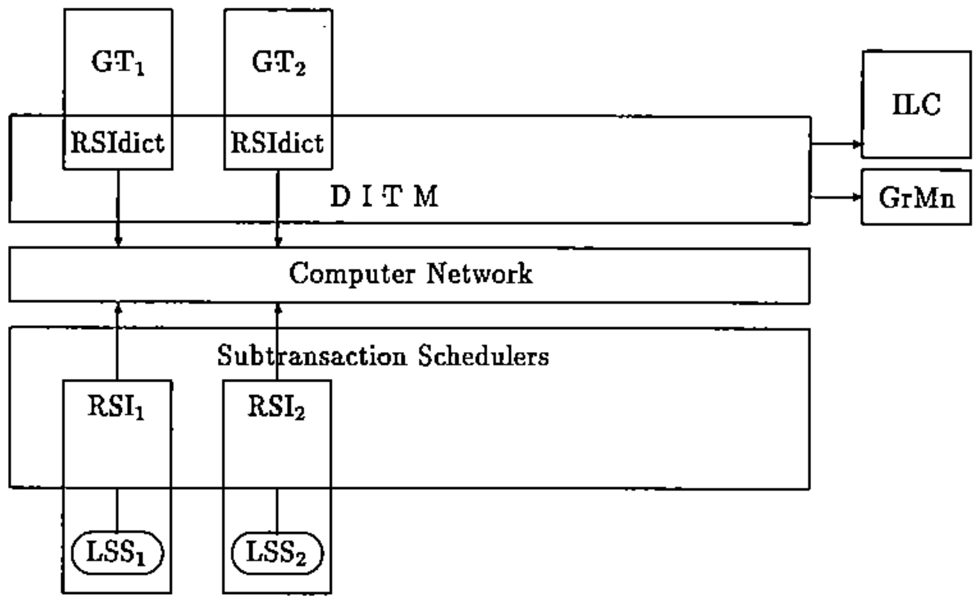

Figure 2. The Interbase System

The major advantage of this architecture is its decentralization feature. The DITM is distributed on all the machines from which IPL programs are executed; that is, each global transaction is associated with an image of DITM. Only the Group Manager of DCC must be run on specific machines. To increase system reliability, the Group Manager consists of a primary and a backup group manager. Upon receiving a starting request from a global transaction, the Primary Group Manager decides the transaction group for the transaction, according to a graph algorithm described in [5]; The Subtransaction Schedulers of the DCC in the individual RSIs guarantee that transactions are executed in quasi serialization order on each LSS. The RSI executes the subtransactions sent to it, in the order specified by the group manager.

The information exchanged within the Interbase System is performed via computer network, and therefore each component of the InterBase System has location transparency.

Each component of InterBase maintains a write-ahead log to keep track of its execution; thus whenever a component of InterBase fails, InterBase can always recover the component to its state right before failure. The Primary Group Manager also monitors the execution of each component of InterBase, such as RSIs and the Backup Group Manager, and recovers the failed ones. The Backup Group Manager monitors the execution of the Primary Group Manager. If the Primary Group Manager fails, the Backup Group Manager sends broadcast messages to all images of DITM and takes over the control of the Primary Group Manager. At the same time, a new Backup Group Manager starts. This feature increases the reliability of InterBase, and also maintains minimum communication costs.

5 Conclusion

This paper has addressed the problems inherent in an environment consisting of distributed, heterogeneous and autonamous software systems. This environment typically arises in the process of fulfilling diverse computational and information processingrequirements.

We presented in this paper the components of Interbase (a project implement at Purdue University), and described how a rule-based approach can be used to organize and resolve inconsistency between different objects in the interbase system. The Interbase Knowledge Base provides for the correctness of IPL programs and security policies, and may address different concepts of interdependent data.

We have also presented the Interbase Logic Controller (ILC). The ILC is used to control the different tasks to be performed within the Interbase environment. The InterBase Parrallel Language supports a powerful description of advanced transactions and provides communication among subtransactions with global transactions.

References

[1] S. G. Akl and D. E. Denning. *Checking classification constraints for consistency and completeness.* In Proceedings if the IEEE Symposium on Security and Privacy, April 1987.

[2] D. E. Bell and LaPadula. *Secure Computer Systems: A unified exposition and multics interpretation.* Technical Report ESD-TR-75-306, MITRE Corporation Bedford Mass., March 1976.

[3] A. Sheth and P. Krishnamurthy, "*Redundent data management in bellcore and bcc data databases.* Technical Report TM-STS-015011,Bellcore Technical memorandum, 1989

[4] A. Elmagarmid, J. Chen, W. Du, O. Bukhres, and R. Pezzoli. *InterBase : An Execution Environment for Global Applications over Distributed, Autonomous, and Heterogeneous Software Systems.* Technical Report CSD-TR-92-016, Department of Computer Sciences, Purdue University, March 1992. (submitted to Computer Magazine).

[5] W. Du, A. Elmagarmid, and W. Kim. *Maintaining Quasi Serializability in Multidatabase Systems.* Proceedings of the 7th Intl. Conf. on Data Engineering, pages 360–367, Kobe, Japan, April 1991.

[6] Y. Leu, A. Elmagarmid and N. Boudriga, *Specification and Execution of Transactions for Advanced Database Applications*, Information systems Journ., vol , 1992

[7] P. K. Chrysantis and K. Ramamrithan, *ACTA: The Saga Continues*, IN Database Transaction Models for Advanced Applications, A. Elmagarmid (ed.), Moragan Kaufmann, 1992

[8] J. Gray, *The Transaction Concepts: Virtues and Limitations*, Proceedings of the 7th Conference on VLDB, pp144-154, 1981