# Multidatabase Interoperability

Y. Breitbart†
Computer Science Department
University of Kentucky
Lexington, KY 40506

**Abstract**

The main purpose of this paper is to provide a brief review of the most current work in the area of multidatabases. We first define the problem and argue that multidatabase research will become increasingly important in the incoming years. We then outline basic research issues in this area and concentrate on issues related to schema integration and semantic heterogeneity, and multidatabase transaction management. The review is not intended to be comprehensive and in spite of our effort to remain objective in selecting review topics, probably reflects the author's biases to some extent.

## 1. Introduction

Recent progress in communication and database technologies has drastically changed user data processing capabilities. The present data processing situation is characterized by a growing number of applications that require accessing and manipulating data from various *pre-existing* database sources located in *heterogeneous* hardware and software environments distributed among nodes of the computer network.

The data sources are *pre-existing* in the sense that they were created independently, in an uncoordinated way without a consideration that one day they may need to be integrated. The DBMSs involved are *heterogeneous* in the sense that they may use different underlying data models, different data definition and data manipulation facilities, and operate in different operating environments. Semantically identical data contained in heterogeneous data sources may have different physical and logical data representation and even different data values.

There are two possible solutions for this problem. The first is physical integration of all data needed by an application into one database. This solution, however, suffers from three major drawbacks: (1) it is expensive, (2) it does not allow maintaining data independently, (3) it leads to unnecessary data duplication, and (4) it requires an expensive application conversion. In addition to that, if an application requires data from external data sources, the physical integration seems to be impossible.

The alternative solution is logical integration of all data needed by an application into one logical database. Such integration creates an illusion of a single database system and hides from users the intricacies of different DBMSs and different access methods. It provides users with uniform access to data contained in various databases, without migrating the data to a new database, and without requiring the users to know either the location, or the characteristics of different databases and their corresponding DBMSs.

The latter approach generated significant interest in research issues related to the logical integration of information into what appears to be a single database. Research on systems facilitating such integration (called multidatabase systems) has been active for about a decade.

Hammer and McLeod [HAMM80] first discussed a system architecture for data access from heterogeneous databases. They proposed a notion of *federated databases* to refer to a collection of independent and pre-existing databases that agreed to cooperate. In 1982, a multidatabase prototype based on this architecture was developed in INRIA [LITW82].

Independent of this work, Computer Corporation of America designed and developed another prototype system - MULTIBASE [LAND82] that was based on a different principle - *global schema* (more on this in Section 2). These two approaches proved to be fundamental to most further work on logical data integration of heterogeneous data sources and data access from such sources.

Data update issues in multidatabase systems were first discussed by Gligor and Popescu-Zeletin [GLIG85]. They outlined basic requirements for a transaction management system that assures consistent data update in a heterogeneous database environment and pointed out the inherent difficulties of data update in such systems.

These papers generated significant interest in the database community and started an intensive study of research issues involved in access and manipulation of data from heterogeneous data sources.

We believe that heterogeneous database research will become even more important in the next decade with the advent of scientific and CAD/CAM database applications. In fact, the recent report of the NSF Workshop on Future Directions in DBMS Research named the area of heterogeneous distributed databases as one of the two most important research areas in the 90's.

The main purpose of this paper is to provide a brief review of the most current work on multidatabases. We outline basic research issues in this area and concentrate on issues related to schema integration and semantic data heterogeneity, and multidatabase transaction management. The review is not intended to be comprehensive and in spite of our effort to remain objective in selecting review topics, probably reflects the author's biases to some extent.

In the next section we briefly discuss research topics that the current work on multidatabases is concentrated on. In Section 3 we discuss in more detail several selected papers on schema integration and semantic heterogeneity, and transaction management. Section 4 concludes the paper and outlines some open problems.

The list of references that we provide is far from being complete and many papers that provided the foundation of

† This material is based in part upon work sponsored by the Center for Manufacturing and Robotics of the University of Kentucky, by National Science Foundation under grants No. IRI-8904932 and No. RRI-8610671, Commonwealth of Kentucky EPSCoR program, and by the Amoco Production Company - Research Center.

multidatabase research are not listed at all. We recognize this deficiency and do not pretend that the references provide a complete picture of what is happening in the area. Due to space limitations, we elected to include only very few references with an understanding that further references may be found in the listed papers. We also did not provide a meaningful discussion of many other multidatabase research issues that were extensively discussed in the literature. Among those are: object-oriented approach to database integration, and query optimization. We refer the interested reader to [POSI89] for further discussion of these topics.

## 2. Current Research Issues

In this section we briefly outline major areas of current research on multidatabases. Most of the recent work on multidatabases has been concentrated in the following areas: (1) schema integration and semantic heterogeneity, (2) transaction management, (3) query optimization, and (4) object-oriented multidatabases.

### 2.1. Schema Integration and Semantic Heterogeneity

A schema integration problem in a heterogeneous database environment can be posed as follows:

*For a given set of local database schemas, create an integrated schema in such a way that each local schema can be considered as a view of the integrated schema.*

There are two basic approaches to this problem. One requires a database administrator to create a global schema for a set of local databases being integrated and each user's application is provided with its own view of the global schema. This is a global schema approach. It is very similar to the design of a conceptual schema for a set of applications in a single DBMS environment, where each application in such environment is provided with its own view of data. In fact, some methods that were developed for the latter problem are applicable to the former. A comprehensive review of methodologies developed for conceptual schema design is given in [BATI86].

On the other hand, there are significant differences between these two problems. One is that in the multidatabase case a global schema generates a *virtual* database or a logical view of the integrated databases, that does not contain any data. Another one is the presence of semantic data inconsistencies in heterogeneous database environment.

In the heterogeneous environment one of the major problems in schema integration is to resolve data inconsistencies that may exist in different data sources for semantically identical data. The most significant problems of such nature are as follows.

1. resolution of naming conventions and naming conflicts that occur when semantically identical data items are named differently or semantically different data items are named identically.
2. resolution of data representation conflicts that occur when semantically identical data items are represented differently in different data sources (for example, data represented as a character string in one database maybe represented as a real number in the other database).
3. resolution of differences in data structures in different data sources.
4. resolution of data scaling conflicts that occur when semantically identical data items stored in different databases using different units of measure.
5. resolution of missing or conflicting data values that occur when semantically identical data items may have some attribute values different or missing in some data sources.

These problems are called domain mismatch problems.

The global schema approach was first described by Dayal and Hwang [DAYA84]. Their method uses a functional data model with two basic constructs: entities and functions. Functions are either single or multivalued. A generalization operator is used to construct new entities from the entities of original databases.

Dayal and Hwang proposed solutions for problems (1) - (3) that involved data renaming, logical data restructuring, and scale conversions. To solve problems (4) - (5) the authors did not try to perform a semantic data analysis to decide which data value over mismatched domains should be presented to the user. They rather made strong assumptions about the domains of the data involved. Their method was used in MULTIBASE [LAND82]. An alternative global schema design method using a functional data model was developed by Motro [MOTR87] (see review of [MOTR87] in the next section).

An alternative approach to schema integration does not require the creation of a global schema. For each application, the database administrator creates a schema describing only data that the application may access in the local database. This schema is called an import schema. The database administrator at each local site also creates a schema of the data that the database has agreed to share with other local sites. This approach is known as a federated database approach [HAMM80].

The federated database approach proved to be very popular and in fact, almost all currently developed multidatabase systems use a federated approach for schema integration. We believe that the federated approach is more advantageous than the global schema approach. It is difficult (if not impossible) to create a single global schema for a large number (potentially thousands!) of data sources. With the global schema approach a single semantic is used to resolve data conflicts for semantically identical data items appearing in several data sources integrated by the schema. This is not the case with the federated approach. A federated approach allows the database administrator to supply users with different data values of semantically identical data items located in different data sources. In the next section, we review in more detail the MRDSM system [LITW86] that is based on a federated approach.

In both cases, schemas that are created for applications as well as a global schema are stored in a multidatabase directory for further reference. With a global schema approach, users of one global schema cannot issue requests against views of the other global schema, while in a federated approach users can refer their queries against each other's schemas.

In almost all multidatabase system prototypes, domain mismatch problems (1) - (5) were resolved straightforwardly by using either data conversions or domain mappings defined by the database administrator. The mappings between domains did not include many-to-many mappings to resolve data inconsistencies. In [DEMI89], a domain mismatch problem is formally defined for the first time, and a solution is

described that encompasses any type of domain matching. A review of [DEMI89] is given in the next section.

Recent work [POSI89] tends to apply a mix of both global and federated approaches to the schema integration. Schema integration and semantic heterogeneity is probably the most active area of research in multidatabase systems. Still many difficult problems remain unresolved. Among them are: automated tools for schema integration and resolving semantic inconsistencies, mapping algorithms for various data models, and application of knowledge based techniques to a schema integration process.

### 2.2. Transaction Management

The major task of transaction management in a multidatabase environment is as follows:

*Ensuring the global consistency and freedom from deadlocks of the multidatabase system in the presence of local transactions (i.e. transactions executed outside of the multidatabase system control) and in the face of the inability of local DBMSs to coordinate execution of multidatabase transactions (called global transaction), under the assumption that no design changes are allowed in local DBMSs*

The difficulties of this task stem from a requirement that each local DBMS operate autonomously and that local transactions are permitted to execute outside of the multidatabase system control. The problem was extensively studied in two basic directions: restricted autonomy of the local DBMS's [PU87] and complete preservation of local DBMS autonomy ([ALON87], [DU89] and [BREI88]).

Restricted autonomy [PU87] implies that the local DBMSs can be modified to enable them to share their local control information (for example, local schedules). This assumption, however, requires design changes in local DBMSs and, as a result, reduces the multidatabase transaction management problem to the identical problem in the homogeneous distributed database environment with hierarchical organization of local sites. This issue has been studied extensively in the literature and is fairly well understood. On the other hand, the study of restricted autonomy is important in that it can determine the minimal information the local DBMSs need to share with a multidatabase system to ensure correct execution of local and global transactions. Pu [PU87] provides some results in that direction but far more work is needed.

A complete preservation of local DBMSs' autonomy implies that local DBMSs cannot be modified in such a way that they can share their local control information. In [BREI88] we considered serializability as a correctness criterion for a multidatabase concurrency control mechanism, and defined a multidatabase transaction management protocol that ensures both global serializability and freedom from global deadlocks, provided that each local DBMS ensures local serializability and freedom from local deadlocks.

Du and Elmagarmid [DU89] argued that serializability might be too strong a requirement in a multidatabase environment and as a result introduced the notion of *quasi serializability*, and proposed quasi serializability as a correctness criterion for a multidatabase transaction management (see review of [DU89] in the next section).

There is, however, no satisfactory transaction management algorithm for a multidatabase system without any restrictions on local DBMSs. One way to find such an algorithm is to impose some restrictions on either the type of global transactions that may be executed in a multidatabase system, or the structure of the local concurrency control mechanisms.

In [BREI90], the latter approach was followed by requiring each local DBMS use the strict two-phase locking protocol (i.e. a transaction may not receive any lock after it releases at least one of its locks, and a transaction releases its locks only after it commits or aborts). A multidatabase concurrency control algorithm was proposed there that ensures both global serializability and freedom from global deadlocks in the presence of failures and for any combination of local transactions, provided that each local DBMS ensures freedom from local deadlocks (see review of [BREI90] in the next section).

It is widely believed that if all the local DBMSs of a multidatabase system would use the strict two-phase locking protocol and would be designed in such a way that they can cooperate in the two-phase commit protocol (for example, each local DBMS would have a prepare-to-commit statement available to users), then the problem of transaction management in such multidatabase system would be trivially solved, even in the presence of failures.

Indeed, if a transaction is in a *prepare-to-commit* state at sites $S_1$ and $S_2$ and the transaction failed at site $S_1$ before it managed to commit, while at site $S_2$ the transaction has committed, then the local DBMS at site $S_1$ would continue to keep the transaction's local locks until the transaction is restarted at site $S_1$ and completes the commit. As a consequence no global inconsistency should occur.

There are, however, three basic problems with this argument in a multidatabase environment:

1. We are not aware of any major commercially available DBMS (with the notable exception of Sybase) that provides users with a *prepare-to-commit* operation. Even, if in the future such a command should become available from a majority of DBMS vendors, it hardly can be expected that every commercially available DBMS would provide such a command.
2. A local DBMS would not know how long it should keep locks for a transaction that failed in the *prepare-to-commit* state. Moreover, the local DBMS is not able to obtain such information by consulting either other local DBMSs involved in the transaction execution, or the multidatabase system. This is because local DBMS is autonomous and is not aware of either other DBMSs involved in the execution of the failed transaction, or of the multidatabase system.
3. The third problem relates to local DBMS security. A global transaction at a local site is a process that is known to the local DBMS under the process identification assigned by the local site's operating system. As soon as this process fails, the operating system of the local site may reassign the failed process identification to some other process of a different transaction. Since the DBMS knows each process by its process identification, the new process with the same process identification will be able to access all data of the failed process and "impersonate" the failed process.

It is interesting to observe that in the case of a homogeneous distributed database the failure of a transaction to commit at a local site is considered a site failure, while in a heterogeneous environment the transaction process at the site

is independent of the local DBMS, and is managed by the local operating system. In the homogeneous distributed database system case, after the site recovers from the failure, the local DBMS would consult either coordinator or other sites as to what decision was made concerning transactions in the *prepare-to-commit* state at the time of the failure.

Research on multidatabase transaction management has just started and only very few results are obtained. Results obtained so far indicate that correctness criteria used for centralized and homogeneous distributed databases could be very hard to enforce in a multidatabase environment. A great deal more work is needed to better understand the problem and to come up with practically viable and provable correct transaction management algorithms in such an environment.

### 2.3. Query Optimization

The query optimization problem in homogeneous centralized and distributed database environments was extensively studied and many outstanding results have appeared in the literature. The problem in a heterogeneous database environment, however, have been studied to a much lesser extent. One of the main difficulties in performing a successful query optimization in such an environment is the inability of the multidatabase system to generate a realistic cost estimate model, since neither local DBMSs, nor a communication system can supply the multidatabase system with accurate cost values.

A query optimization problem in a multidatabase environment was first addressed in MULTIBASE [LAND82]. The MULTIBASE's algorithm performs a query optimization at two levels: global and local. Global optimization refers to subdividing of a user query into a set of subqueries to be sent to local sites for processing. A set of subqueries and a set of operations to generate a response to the original query can have a significant impact on the cost of query execution. Minimizing these costs is a task of a global optimizer. Local optimization refers to a generation of a strategy to execute a single subquery at a local site with minimal costs.

Many multidatabase prototype systems include a query optimizer but not many have been described. At this time the research in multidatabase query optimization is at a very early stage.

### 2.4. Object-Oriented Multidatabases

Research in object-oriented databases has just started and very few papers have appeared so far. At the NSF Workshop on Heterogeneous Databases the topic was discussed [POSI89] at some length. We believe that potentially, the object-oriented multidatabases may be very important and would contribute to a solution of domain mismatch, transaction management, and query optimization problems in a multidatabase environment. However, at this point, much more research is needed to evaluate the benefits of this approach.

## 3. Review of the Current Work

In this section, we review 3 papers ([LITW86], [MOTR87], and [DEMI89]) related to schema integration and semantic heterogeneity and 2 papers ([DU89], [BREI90]) related to transaction management in multidatabase systems.

Litwin and Abdellatif [LITW86] describe a multidatabase system based on a federated architecture, while Motro [MOTR87] provides a formal description of a system based on a global schema approach (called superviews in [MOTR87]). DeMichiel [DEMI89] describes a method to resolve semantic inconsistencies for the case of mismatched domains in databases that are to be integrated. Du and Elmagarmid [DU89] introduce a new notion of correctness for a concurrency control in a multidatabase environment and provide a theoretical analysis of it. Breitbart, Silberschatz and Thompson [BREI90] describe a fault-tolerant transaction management algorithm in a multidatabase environment where each constituent DBMS uses the strict two-phase locking protocol.

#### Litwin and Abdellatif [LITW86]

MRDSM allows integration of various relational database systems. The integration is performed largely at two levels: database definition of relations being integrated described in the MRDSM data definition language, and the multidatabase data manipulation language available to users of the system. The database definitions of the integrated view of cooperating databases are created dynamically by a database administrator and either exist for the duration of the user's session with MRDSM or are stored in the directory for further use.

The data manipulation language allows users to perform the joining of data in different databases, the dynamic transforming of actual attribute values into user-defined value types, data retrieval from different databases in the same query, and data aggregation of data from different databases using various built-in functions.

The data definition language includes capabilities to define export schemas [HAMM80] and define user access rights. For a collection of databases being integrated by MRDSM, the database administrator assigns a unique name called a multidatabase name.

On a set of relations integrated by MRDSM, the database administrator may define three types of dependencies: manipulation, privacy, and equivalence dependencies. A manipulation dependency triggers processing in one database when processing is conducted in another database. As an example, an insertion of a tuple in some relation may trigger insertion of the same tuple into some other relation. A privacy dependency triggers a check of the user's rights for the database to be accessed. An equivalence dependency identifies for each database to be accessed its primary or candidate keys. Equality of primary keys from two separate relations corresponds to the same real object information stored in these relations. Since users are required to specify from what relations to select attribute values for a tuple identified by the same primary key in two different relations, the problem of mismatched attribute values for the same object is not addressed.

The query and data manipulation languages of MRDSM are based on a tuple calculus and patterned after QUEL. It includes retrieve, modify, store, delete, copy, move, and replace statements. To manipulate one or more databases a user is required to submit an open command, where each database to be manipulated is listed along with the access mode (shared or exclusive). To facilitate access of several databases in one user query, several new concepts are proposed in MRDSM. Among them are: multiple identifiers, and semantic variables.

A multiple identifier refers to the same relation name at two different sites. It is assumed that if at two different local sites two relations have the same name, then they contain semantically related information. For example, if a relation at one site named restaurant and a relation at the other site also is called restaurant, then both relations presumably contain

information (not necessarily the same) about restaurants. A semantic variable allows users to identify several relations that contain semantically relevant information with one name in the user query.

Users of MRDSM are able to define dynamic attributes for either the duration of the query or the duration of the user's session with MRDSM. A dynamic attribute is a function defined on actual attributes that the user is authorized to access. Finally, using commands copy and move, relations from one site can be copied or moved to another site. Users may create a relation as a result of their query and move it to an indicated data location.

In summary, MRDSM is a heterogeneous database system that is capable of integrating pre-existing relational databases. This system allows users to join data from different relations, to define dynamic attributes, and to dynamically aggregate data from different relations. From this view point, it is the most complete prototype built so far. On the other hand, MRDSM does not deal with physical data distribution, nor with the issues of query optimization in the heterogeneous environment. It is also not clear whether new ideas in MRDSM (such as semantic variables and multiple identifiers) can be extended to other than relational databases.

**Motro [MOTR87]**

In contrast with the MRDSM system that is based on a federated approach, the superview approach [MOTR87] is based on a global schema architecture approach. The superview approach generates a *virtual* database as a pair *<schema, mapping>*, where mapping is defined from a global schema into actual database schemas being integrated.

Two tools are described in [MOTR87]: *a virtual database generator or a data definition language processor*, and *a virtual query processor*. A virtual database generator is an interactive program that generates a virtual database from the schemas of actual databases being integrated and integration statements of the data definition language defined in the paper. A mapping generated by a virtual database generator is stored as a new virtual database for future reference. A virtual query processor using mappings stored for a virtual database translates a query submitted to a virtual database into a query or a set of queries against each individual database being integrated by a virtual database.

Unlike MRDSM which uses a relational database model for an integrated schema, a functional database model is used for an integrated schema in [MOTR87]. Each database in the model is a set of classes, and two relationships are defined on these classes: Att and Gen. Att relates one class as being an attribute of the other one, and Gen relates one class as being a generalization of the other one. Each relation is a class but not vice versa. Therefore, the superview approach is capable of integrating other than relational databases.

Each class has a key that is defined as a subset of its attributes. For each class, key values uniquely determine values of other attributes of the class. Several keys may exist for an class. Keys convey important semantic information, namely, two classes that have the same keys are considered to be semantically related and therefore can be integrated. On the other hand, if two classes are keyed on the same set of attributes and these classes contain some other (non-key) attributes in common, then their common attributes should contain the same values for the same key values in both classes. This means that no semantic inconsistency is allowed. It appears, however, that renaming common non key attributes in classes with a common key would place a problem of a semantic inconsistency resolution in the hands of the database administrator who defines a virtual database for an application.

The data definition language contains a variety of operators that allow the generation of new classes and the destruction of some other classes. The major operators of the language are: *meet*, *join*, *aggregate*, *rename*, and *telescope*. Operation *meet (join)* for any two classes generates a third class that is their intersection (union). For example, for relations *Person(SSN#, NAME, ADDRESS)* and *Student(SSN#, NAME, RANK, ADVISOR)*, *meet(Person, Student)* = *(SSN#, NAME)* and *join(Person, Student)* = *(SSN#, NAME, ADDRESS, RANK, ADVISOR)*. Operation *meet* creates a new class that is a generalization of classes that are arguments of *meet*. Operation *join*, on the other hand, generates a new class $S$ such that each class that is an argument of *join* is a generalization of $S$.

Operation *fold* enables a generalization class to include attributes of a generalized class. Operation *aggregate* creates an intermediate class $C$ from a subset of attributes of class $S$ and $C$ becomes an attribute of $S$. Operation *telescope* is the inverse of operation *aggregate*. Operation *rename* allows the assignment of a new name to a class or its attribute. These operations are used to create a virtual database from the local database schemas.

A user submits a query to a superview. The query is translated over each of the integration operators that were used in the superview definition. The result of the translation is a set of queries to local schemas. Each of the queries from that set is executed by a local system and results are passed back to the query processor. The query processor then combines results obtained from local sites using the definitions of the superview. If a requested value cannot be found in a local database, then a fixed semantic proposed in the paper indicates what data value is sent back to the user.

In summary, the proposed formal system enables one to formalize a process of schema integration in a heterogeneous environment. This approach can be useful in designing tools for automating the schema integration process.

**DeMichiel [DEMI89]**

In both systems ([LITW86], [MOTR87]) the problems of resolving semantic inconsistencies were handled by the database administrator during the creation of the integrated view for the application. In [DEMI89], a formal system is presented that allows the derivation of an attribute value for mismatched domains.

DeMichiel addresses the domain mismatch problem for multidatabases with each constituent DBMS using a relational data model. The approach taken in [DEMI89] makes use of two mechanisms: *domain mapping* and *virtual attributes*.

A *virtual attribute* is similar to a real attribute in that it denotes a property of some data item. However, it does not exist physically and its values are logically derived from actual attributes or from other information in the database. A *domain mapping* defines a correspondence between domains of two different attributes (one or both attributes can be virtual). Domain mappings and virtual attribute definitions are stored in the multidatabase directory as a part of an integration schema in a federated or a global schema integration model.

If two domain values for the semantically related attributes in two relations are mismatched, the following steps are

taken: (1) real attributes are mapped to virtual attributes with a common domain to resolve domain differences, and resulting relations would then become compatible for executing various relational operations, (2) relational operations are performed on resulting relations, (3) a query is evaluated using the relations resulting from operations applied at step (2), or relational operations are repeated for relations obtained from previous application of step (2), and finally (4) final results are presented to users. The paper contains an example that fully illustrates the described process.

Attribute values are extended to contain a *set* of atomic values. The semantic of this is that any value from the set may actually be an attribute value. An attribute value that contains a set of atomic values is called *partial* value. A tuple is called *definite* if each of its attribute values is atomic, otherwise a tuple is called indefinite. A relation is called *total* if each of its tuples is definite, otherwise it is called *partial*.

Original relations that are being integrated are always definite. They may, however, contain mismatched attributes. A domain matching operation, however, may generate virtual attributes with partial values. As an example, consider two relations that contain the attribute *ADDRESS*. In one relation the attribute may contain street address while in the other it may contain only city name. After both attributes are matched to a virtual attribute *ADDRESS*, both relations may contain, then, partial values for the virtual attribute *ADDRESS* (if, for example, two different cities have the same street).

Each of the original relations is in the third normal form. Two relations $r$ and $s$ are consistent if they have the same relational schema, the same key attributes, and for any two tuples from $r$ and $s$, if these tuples have the same values for their key attributes, then for any non-key attribute $C$, $r.C$ and $s.C$ have at least one value in common. Extended union operation is defined for union compatible and consistent relations.

Extended relational operations such as union, join, select, and project allow the generation of new relations from original relations in such a way that information about the virtual attribute may become definite. Each operation may produce either *true* or *maybe* values, where the *true* value is defined as a value that the attribute must contain, and the *maybe* value is defined as a value of the attribute that cannot be excluded from the result.

It appears that the proposed approach is the first formal method that enables a multidatabase system to generate a unique attribute value even when attributes are mismatched in the integrated relations. On the other hand, the developed approach is applicable only to relational data sources. It is clear that further work is needed to extend the results from [DEMI89] to other database models.

**Du and Elmagarmid [DU89]**

Du and Elmagarmid [DU89] proposed to relax the requirement of serializability as a correctness criterion for concurrency control mechanisms of a multidatabase system. They introduced a new notion of correctness: *quasi serializability*.

A local schedule is a sequence of operations from both local and global transactions executed at the local site. A global schedule is a collection of local schedules. A global schedule is *quasi serial* if and only if each local schedule is conflict serializable and there is a total order of global transactions such that for any two global transactions $T_i$ and $T_j$ if $T_i$ precedes $T_j$ in the total order, then all $T_i$ operations precede all $T_j$ operations in all local schedules in which both appear. A global schedule is *quasi serializable* if it is conflict equivalent to a *quasi serial* schedule.

Du and Elmagarmid prove in [DU89] that a global schedule is *quasi serializable* if and only if a *quasi serializability* graph of the schedule is acyclic, where *quasi serializability* graph $G$ of a schedule $S$ is defined as follows.

Nodes of $G$ are global transactions and $G$ contains an edge $<T_i, T_j>$ if and only if $T_i$ conflicts $T_j$ in the same local schedule or there is a set of operations $o_i(x)$, $o_1(x)$, $o_1(y)$, $o_2(y)$, ..., $o_k(t)$, $o_k(z)$, $o_j(z)$ and a local schedule $L$, such that

$$L = ...o_i(x), ..., o_1(x), ..., o_k(t), ..., o_k(z), ..., o_j(z), \cdots$$

In order to use *quasi serializability* as a correctness criterion in a multidatabase environment, it is assumed in [DU89] that there is no data dependency between data items located at different local sites processed by the same global transaction. For example, a global transaction defined as follows: $a = a + b$, where $a$ and $b$ are located at different local sites, would have data dependency between $a$ and $b$. In contrast, for a transaction $a = 30$; $b = 20$, where again $a$ and $b$ are located at different local sites there is no data dependency between $a$ and $b$.

Every conflict serializable schedule is also quasi serializable but not vice versa. In fact, in [DU89] it is shown that sets of view serializable and quasi serializable schedules are incomparable. It is also shown there that the transaction management algorithm proposed in [ALON87] generates quasi serializable schedules.

Quasi serializability represents a new and interesting notion of global database consistency that is weaker than serializability. It is applicable to such global transactions that can be executed in parallel at local sites (since they satisfy the data independency condition). It is however, not clear whether there is an effective algorithm that verifies that a global transaction does satisfy the data independency criterion.

**Breitbart, Silberschatz, and Thompson [BREI90]**

In [BREI90] a different approach to global database consistency is taken. It is assumed there that each local DBMS uses the strict two-phase locking protocol and global serializability is used as a criterion of global database consistency. The two phase commit protocol is used to perform a global transaction *commit* operation. However, no assumption is made that local DBMSs make a *prepared* state of the transaction visible to a multidatabase system. Failures may occur at any time during the transaction execution. Major failures considered in [BREI90] are: transaction failures, system failures, site failures, communication failures, and failures of global subtransaction at a local site without a site failure.

Let us assume that a failure occurs during the processing of a *commit* operation of a global transaction. Consider the case where the multidatabase system decides to commit transaction $T$. Suppose that $T$ at site $S$ fails without having the appropriate local DBMS log records in stable storage and before $T$ at site $S$ has received the commit message from the system. If such a failure occurs, then $T$ must be undone at $S$. However, the multidatabase system considers $T$ to be committed and thus, the MDBS must redo $T$ at $S$. As far as the local DBMS is concerned, redoing the transaction constitutes a new transaction at that site, without any connection to the

failed one. Thus, it is possible, that between the time that $T$ is restarted at $S$ after the failure and the time that the restarted transaction $T$ at site $S$ obtains the local write locks from the local DBMS to redo the $T$'s *write* operations, the local DBMS may execute some other local transactions that, in turn, may lead to the loss of global database consistency.

The above example illustrates major difficulties in the design of a fault-tolerant transaction management algorithm. To overcome the difficulties, in [BREI90] additional restrictions on a set of global data items were imposed. All global data items are subdivided into two mutually exclusive classes: globally and locally updatable. Any global transaction that contains write operations may write only globally updatable data items and may not read locally updatable data items.

The separation of global data items into two classes generally does reflect the current data processing situation in major corporations, where each data item can be updated only by one procedure. On the other hand, disallowing global update transactions to read locally updatable data items is a rather strong limitation on local autonomy.

Under these restrictions the authors propose a fault-tolerant concurrency control algorithm. Their algorithm uses a *commit* graph that is defined as follows. The nodes of the graph are global transactions and local sites. An edge $<T, S>$ is created if and only if transaction $T$ is being committed at $S$. A concurrency control algorithm that submits global transactions' *commit* operations in such a way that there is no loop in the *commit* graph ensures global serializability for any system of local transactions and any combination of failures of described types.

In a multidatabase environment with each local DBMS using the strict two-phase locking protocol, the problem of global deadlocks may occur. A global deadlock occurs if every local DBMS is not in a deadlock, but an union of local wait-for-graphs contains a loop. In the paper the authors present a deadlock-free recovery manager algorithm and prove its correctness.

Algorithms presented in [BREI90] fully describe a transaction management system in a multidatabase environment without requiring any modifications of local DBMSs. Many of today's commercially available DBMSs are using the strict two-phase locking protocol. This means that the proposed algorithms can be employed in the current data processing environment (and, in fact, they are being implemented in the ADDS project). Even, if some day local DBMSs will make the *prepared* state of the commit process visible to the multidatabase system, then the algorithms proposed in the paper (or similar ones) still will be needed to ensure global database consistency and freedom from global deadlocks.

## 4. Conclusions

In this paper we briefly surveyed two areas of multidatabase research: schema integration and semantic heterogeneity, and multidatabase transaction management. We also reviewed 5 selected papers in these areas. Multidatabase research is at a very early stage. Even in the areas that we reviewed and that are currently the most active areas in multidatabases, a lot more work needs to be done. We still do not have comprehensive methodologies to build multidatabases and to resolve semantic inconsistencies. We need better understanding of transaction management requirements in a multidatabase environment as well as correctness criteria for transaction management in such environments.

We believe that for multidatabases involving hundreds of local databases, expert systems and knowledge based techniques are required to help users in their access to the data. The work on this topic has yet to start. Query optimization methods for homogeneous systems need to be revisited to determine their applicability to a multidatabase environment. Application design methods for a distributed homogeneous database environment should be revised to include heterogeneous database applications.

It would be much easier to develop future multidatabase systems if operating systems, communication interfaces, and database systems were standardized. But it is utopian to believe that such comprehensive standards will be developed and more importantly will be enforced. Nevertheless, it is important to conduct work on standardization of communication interfaces and transaction management systems. For example, if TCP/IP, SQL, the strict two-phase locking and two-phase commit protocols would be accepted by all vendors, the multidatabase transaction management problem would become much more manageable. We believe that research results in multidatabase systems will provide meaningful input to the standardization effort currently under way.

A heterogeneous data processing environment will be a prevalent environment in the next decade and therefore a need for production level multidatabase systems is obvious. Multidatabase research will play an increasingly important role in building a foundation for such systems to appear.

## References


[ALON87] R. Alonso, H. Garcia-Molina, and K. Salem, "Concurrency Control and Recovery for Global Procedures in Federated Database Systems", IEEE, Data Engineering, September, 1987

[BATI86] C. Batini, M. Lenzerini, S. Navathe, "A Comparative Analysis of Methodologies for Schema Integration, CN Computing Surveys, 18, 4, 1986.

[BREI88] Y. Breitbart, A. Silberschatz, "Multidatabase Update Issues", Proceedings of ACM SIGMOD Conference, 1988.

[BREI90] Y. Breitbart, A. Silberschatz, G. Thompson, "Reliable Transaction Management in a Multidatabase System", Proceedings of ACM SIGMOD Conference, 1990.

[DAYA84] U. Dayal, H.-Y. Hwang, "View Definition and Generalization for Database Integration in a Multidatabase System", IEEE Transaction on Software Engineering, SE-10, 11, 1984.

[DEMI89] L. DeMichiel, "Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains", IEEE Transaction on Knowledge and Database Engineering, 1, 4, 1989.

[DU89] W. Du, A. Elmagarmid, "Quasi Serializability: a Correctness Criterion for Global Database Consistency in InterBase", Proceedings of the International Conference on Very Large Data Bases (VLDB), 1989.

[GLIG85] V. Gligor, R. Popescu-Zeletin, "Concurrency Control Issues in Distributed Heterogeneous Database Management Systems." Distributed Data Sharing Systems. Eds. F. Schreiber and W. Litwin. North-Holland, 1985, 43-56.

[HAMM80] M. Hammer, D. McLeod, "On Database Management System Architecture", in Infotech State of the Art Report, vol. 8: Data Design, Pergamon Infotech Limited, 1980.

[LAND82] Landers, T. and R. Rosenberg. "An Overview of Multibase." Distributed Data Systems. Ed. H. Schneider. North-Holland, 1982, 153-184.

[LITW82] Litwin, W., J. Boudenant, C. Esculier, A. Ferrier, A. Glorieux, J. La Chimia, K. Kabbaj, C. Moulinoux, P. Rolin, and C. Stangret. "SIRIUS Systems for Distributed Data Management." Distributed Data Bases. Ed. H. J. Schneider. North-Holland, 1982.

[LITW86] W. Litwin, A. Abdellatif, "Multidatabase Interoperability", IEEE Computer, 12, 1986.

[MOTR87] A. Motro, "Superviews: Virtual Integration of Multiple Databases", IEEE Transaction on Software Engineering, SE-13, 7, 1987.

[POSI89] Position Papers, 1989 Workshop on Heterogeneous Databases, Sponsored by NSF, December 1989.

[PU87] C. Pu "Superdatabases: Transactions Across Database Boundaries", IEEE Data Engineering, September 1987