1993

The Integration of Database Systems

Tony Schaller

Omran A. Bukhres

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Xiangning Liu

Report Number:
93-046

The Integration of Database Systems

Tony Schaller, Omran A. Bukhres,
Ahmed K. Elmagarmid and Xiangning Liu

CSD-TR-93-046
July 1993

The Integration of Database Systems

Tony Schaller
Molecular Design Ltd.
2132 Farallon Drive
San Leandro, CA 94577

Omran A. Bukhres, Ahmed K. Elmagarmid and Xiangning Liu
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
e_mail: {bukhres,ake,xl}.cs.purdue.edu

1 Introduction

A database system is composed of two elements: a software program, called a database management system, and a set of data, called a database. The data in a database is organized according to some data model, such as the relational model used in a DB2 database [DW88] or the hierarchical model found with IMS databases [Dat77] . Users access the data through an interface (the query language) provided by the database management system. A schema describes the actual data structures and organization within the system.

During the decade of the nineteen-seventies, centralized databases were predominant, but recent innovations in communications and database technologies have engendered a revolution in data processing, giving rise to a new generation of decentralized database systems. Such distributed systems have proved to be well suited to the trend toward corporate decentralization and the development of networking technologies that characterized the nineteen-eighties.

A fundamental distinction must first be drawn between distributed, heterogeneous, and multi-database systems. A distributed database system is made up of a single logical database that is physically distributed across a computer network, together with a distributed database management system that answers consistent queries and updates. A distributed database furthermore implies

homogeneity, in that all its physical components run the same distributed database management system. The distributed database system supports a single data model and query language, with an unambiguous schema. Conversely, a heterogeneous database system is a distributed database system that includes heterogeneous components at the database level; these may include a variety of data models, query languages, schemas, and access heterogeneities. A federated database is synonymous with a heterogeneous database system [HM85]. Generally commercial federated databases have been implemented as tightly coupled within a master schema that fully defines the scope of integration. A major distinguishing factor between distributed and heterogeneous database systems is the implied ability of the component databases to function independently in the latter. Finally, a multidatabase system is a collection of loosely coupled element databases, with no unified schema applied for their integration. Although a typical multidatabase system supports only queries, some also allow updates to element databases. The loosely coupled approach to federated database systems provides a greater degree of autonomy for the component systems, since no central authority is imposed. In addition, loose coupling generally tends to scale better to very large systems, since it is difficult to support a central authority over such systems [BCD+93]. On the other hand, the maintenance of consistency is more difficult in a system that lacks a central authority.

2 The Context of Heterogeneous Systems

Currently, data processing is increasingly characterized by applications that involve accessing and manipulating data from many pre-existing databases. These databases are typically located in heterogeneous and autonomous software and hardware platforms which are distributed over the many sites of a computer network. The distribution of these systems reflects the decentralized nature of modern business, while their heterogeneity and autonomy arise as a consequence of the diversity of corporate computational and information processing requirements. Originally, these systems ran in isolation to support their individual applications, but it has become evident that inter-system cooperation would permit more complex applications involving multiple systems. Unfortunately, the potential for cooperative interaction was not considered in the original system design, and there is as yet no general model supporting interoperability among isolated systems. As

a result, supporting global applications involving multiple systems is a formidable task and remains largely manual.

The unwieldy nature of this current situation is epitomized by the French Teletel System, which provides its 1.8 million users access to more than 1,500 separate databases. Such a plethora of systems demand multiple access methods and user paradigms, presenting a significant obstruct to easy user access. On the other hand, it would not be feasible to require organizations to convert all their pre-existing systems to a single unified standard.

Two possible solutions to this bottleneck may be envisioned. The first would entail the physical integration of all data pertaining to a given application into one database. However, this approach would not only be expensive and complex, but it would also result in unnecessary data redundancies and would prohibit the maintenance of independent databases. Data under the control of independent organizations would in any case remain outside this integration framework.

A more feasible approach involves, the logical integration of all data pertaining to a given application into one logical database, providing the user with the illusion of a single database. Information is integrated from pre-existing heterogeneous local databases in a distributed environment, while global users are presented with transparent methods for accessing all information encompassed by the system.

In general, the integration of distributed heterogeneous database management systems is limited by the following constrains. First, heterogeneities exist at various levels — hardware, operating systems, data models, accessing capabilities, database management systems, and data formats — making it virtually impossible to design a common approach capable of addressing all aspects of heterogeneity. Second, a distributed heterogeneous database management system must integrate existing information systems, each of which was designed independently with little thought to possible inclusion in a larger system. Third, the underlying assumptions and semantics of each existing system must be fully understood, an unlikely prospect in view of the typically limited documentation of these systems. Finally, the integrated system design must strive to create only minimal changes in existing systems. Transactions, whether queries or updates, involve access to individual systems, each with its own set of requirements and standards which must be reconciled

by the distributed heterogeneous database management system.

2.1 Motivation

Organization wide access to data and software resources requires the interconnection of previously isolated applications and systems. An end-user in a heterogeneous computing environment should be able not only to invoke multiple existing application systems but also to coordinate their interactions and associations. In today's computing world, databases are proposed as a solution to the problem of shared access to heterogeneous resources, both software and hardware, that characterize multiple autonomous applications. There are many benefits to integrating pre-existing systems. These benefits are:

- Data Sharing

  -Among Applications

  -Among Users

- Easier Application Development

- Evolutionary System Expansion

- Global Access to Local Data

Organizations typically are made up of a collection of different departments, each making autonomous decisions about its operations within the larger enterprise. A first step toward the integration of such diversified systems is the definition and characterization of the autonomous design of the databases which support these individual information systems. Several contextual issues need to be addressed in the process of arriving at such definition. Widespread heterogeneity arises naturally from a free market of ideas and products, some of which prove to be more widely adapted than others to specific applications. Such diversified market demand will continue to foster heterogeneity in future systems. The tendency toward heterogeneity in large corporations is reinforced by the political decentralization of power and by the frequent acquisition and merger of previously independent companies. Furthermore, the rapid pace of innovation in the computer

field militates against the establishment of the sort of time-tested standard required to create a homogeneous system.

2.2 Cooperative Computing

The necessity of cooperative computing is becoming increasingly evident as we evolve toward large and more heterogeneous computing systems. Cooperative computing describes the ability of two or more programs to communicate or work together to accomplish specific tasks. Frequently, cooperative computing also implies communication between two or more different execution domains, which may range from different run-time support systems with a single processor to physically distinct processors with a distributed computing system.

The need for such cooperative computing arises in many contexts. The nature of an organization determines how and with which other organizations cooperation may be established, as well as the extent to which this cooperation will be achieved through information exchange and processing. For example, organizations whose activity mainly consists of information processing, such as banks, travel agencies, insurance companies, and hospitals, should be especially interested in instituting a cooperative flow of digitized information among their computers. Indeed, such industries as international banking, international air carriers, and travel agencies have already largely automated cooperative information processing activities such as bank transfers and airline and hotel reservations.

The move toward cooperative computing is also strong in international trade. An example of this trend is the emergence of standards and systems for computerized trade shows, such as the Electronic Data Interchange for Administration, Commerce, and Trade (EDIFACT) standard [IS9735] given in the Trade Data Interchange Directory, which is issued and maintained by the United Commission for Europe [Vei90].

3 Aspects of Database Integration

Database integration conceptually combines participating databases to form a single cohesive interoperable multidatabase. Such a multidatabase is capable of providing uniform user access interfaces to the component heterogeneous distributed database systems.

Multidatabase systems combine autonomous and heterogeneous component (or local) database systems into a global database system. In multidatabase systems, global transactions are divided into subtransactions, with one subtransaction per local system that the global transaction accesses. Local transactions may be executed at each local database system. A conceptual view of a multidatabase system is shown in Figure 1. A global transaction G_i, and its decomposition into subtransactions $G_{i,1}$, $G_{i,2}$, ..., $G_{i,n}$ is shown. Also, a local transaction $L_{j,1}$ that executes at $LDBS_1$, and a local transaction $L_{k,n}$ that executes at $LDBS_n$, are shown.

3.1 Commitment in Multidatabase Systems

One of the most difficult problems with implementing reliable transaction management in multidatabase systems is the problem of atomic commitment of global transactions. That is, ensuring that if any of the effects of a global transaction are executed, then all of the effects will be executed. In fact, it has been shown that it is *impossible* to do in general without violating local autonomy [ME91]. The two phase commit protocol (2PC) [EGLT76] has been used for tightly coupled distributed database systems. However, there are at least two problems with applying 2PC to the multidatabase case.

First, current database systems do not generally provide the (visible) prepare-to-commit state which is necessary to implement 2PC. And, if even one component system at which a transaction executes does not support the prepare-to-commit state, it will not be possible to use the 2PC algorithm. Second, even if database systems used in the future generally do provide the prepare-to-commit state, 2PC can severely violate local execution autonomy which we will talk about the next sections. Blocking can often occur with 2PC, so it becomes possible for a remote system to, in effect, lock another system's data for indefinite periods of time.

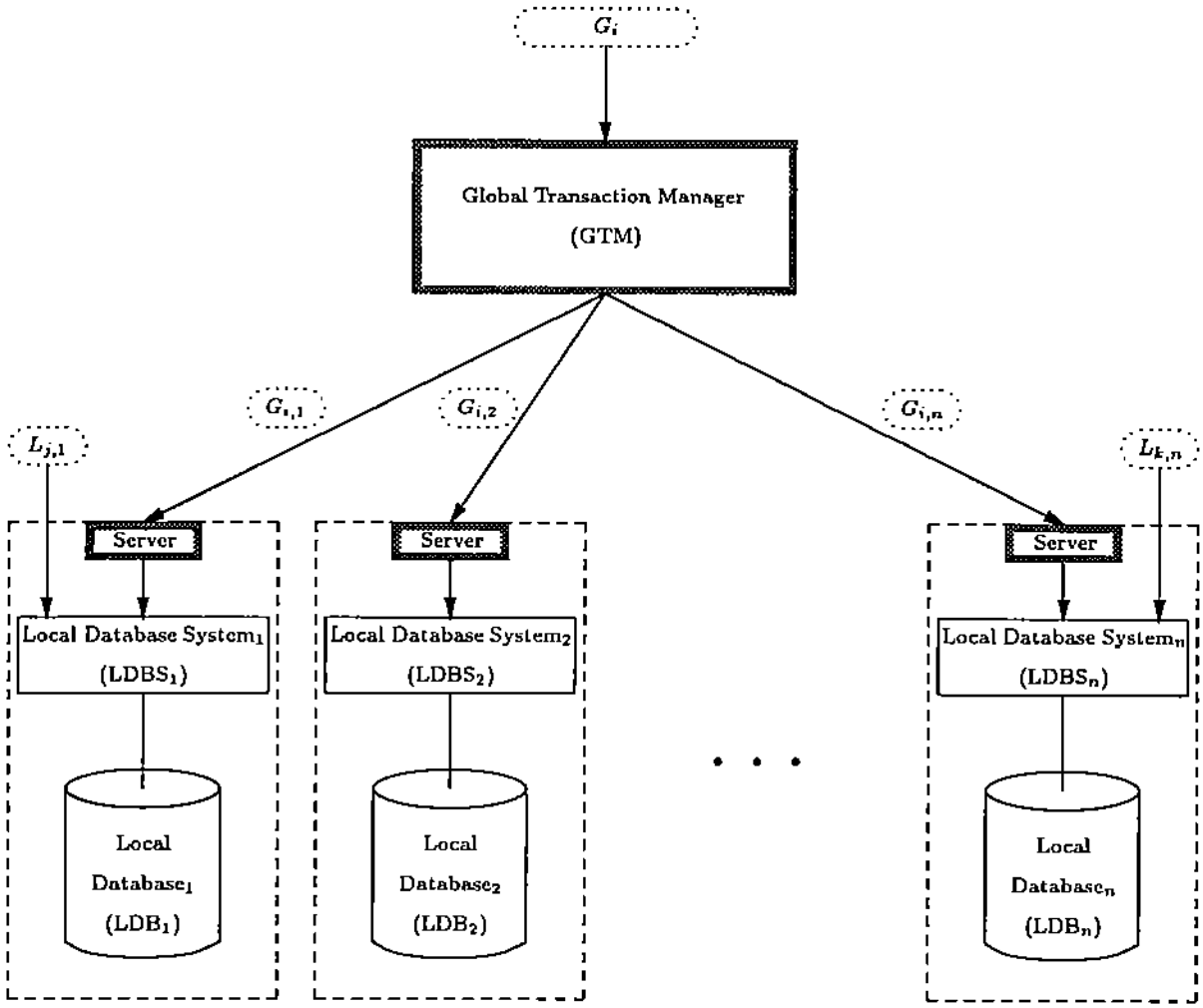

Figure 1: Conceptual Multidatabase Architecture.

One basic approach to handling the atomic commitment problem, that avoids the blocking behavior of two phase commitment, is to use compensating transactions, introduced by Garcia-Molina, such as those used in Sagas [GMK88] or the (multidatabase) Flex transaction model [ELLR90]. Compensating transactions can undo the effects of a committed transaction, so that subtransactions of a transaction can be committed independently. Compensation can be considered an optimistic approach, in the sense that one goes ahead and commits subtransactions, with the hope that the entire global transaction will commit. If it does not, then the committed subtransactions can always be undone. Compensation helps support long-lived transactions, since data is not blocked until the entire transaction commits. In addition, compensation does not require a visible

prepare-to-commit state. However, the applicability of compensation depends on the semantics of the data/transactions involved. So, compensation will not work for all cases. If the operations that can be performed on a data item are not commutative (i.e. if the order of execution matters), then compensation will not work. Also, if the effects of a subtransaction correspond to a real world event (e.g. launching a missile), then the effects might not be compensatable.

3.2 Transparency of Multidatabase Systems

A multidatabase system achieves transparency by separating the higher-level semantics of a system from lower-level implementation and structural issues. Tamer Özsu [ÖV91] has described these transparences, as follows:

- Network transparency. Also referred to as distribution transparency, network transparency, can be considered from the viewpoint of either the services provided by the systems or the data. Network transparency involves both location transparency, in that the physical location of services or data is invisible during high-level applications, and naming transparency, in that a unique name is provided for each object in the system.

- Replication transparency. Data pertaining to a given database may be stored at more than one site. Data replication can improve system reliability and performance and data availability, while high-level applications need not be aware of the redundancy.

- User transparency. Multiple users can access the system simultaneously without mutual awareness or interference. User transparency is created by effective system concurrency control.

- System transparency. The details of varying computer and operating systems and intersite communication mechanisms are screened from user view at the multidatabase level.

- Data semantics transparency. Potential conflicts among data stored in multiple databases should be reduced.

3.3 Characteristics of Database Systems

Database systems are characterized by the following three aspects, as graphically depicted in Figure 2:

- Autonomy. A database can unilaterally formulate its own concurrency control policy and export schema for interacting with other databases. This characteristic may apply to a single database or to multiple databases.
- Heterogeneity. Data models, query languages, and processing methods may differ between multiple databases. As illustrated in Figure 2, such heterogeneity may include systems that are query only, restricted update, non-guaranteed update, and automatic & guaranteed update.
- Distribution. The databases are physically located at different sites. This aspect may include single sites, multiple sites in LANs(Local Area Networks) or WANs(Wide Area Networks), and multiple networks of LAN/WAN.

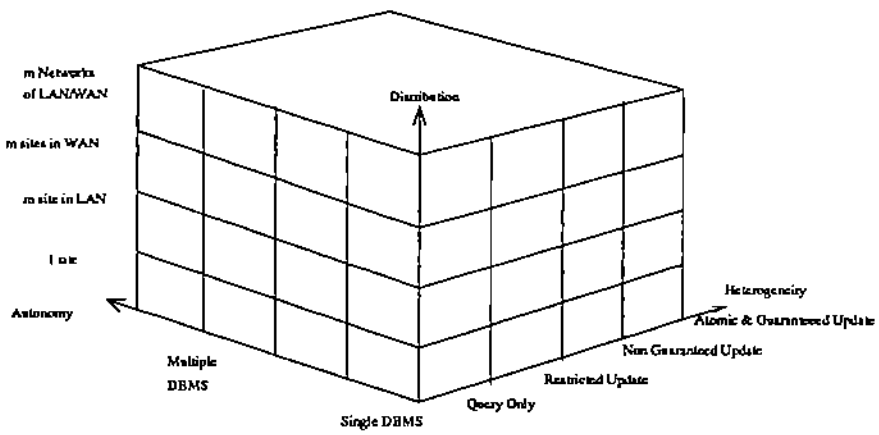

Figure 2: Dimensions of Database Systems

A database can exist at any point in the space defined by the three dimensions of Figure 2. For example, a database may be on multiple sites in a WAN, consist of multiple autonomous DBMSs, and have restricted update.

The integration of database systems must take into consideration not only their characteristics as outlined above, but also the cost and performance of the resulting multidatabase system.

Moreover, we can break the Heterogeneity dimension of the previous figure into three different possible dimensions: Model, Access, and Processing (Figure 3). The model dimension represents the different data models the are being integrated. Therefore, the important task in the integration process then is how to merge together two different databases through the different data models. The access dimension represents the heterogeneous accesses that might exist as a result of the integration. The last dimension is the processing or what we call the heterogeneous execution. This dimension represents the synchronization among the concurrent execution of the global and local transactions while guaranteeing the consistency of the multidatabase system.

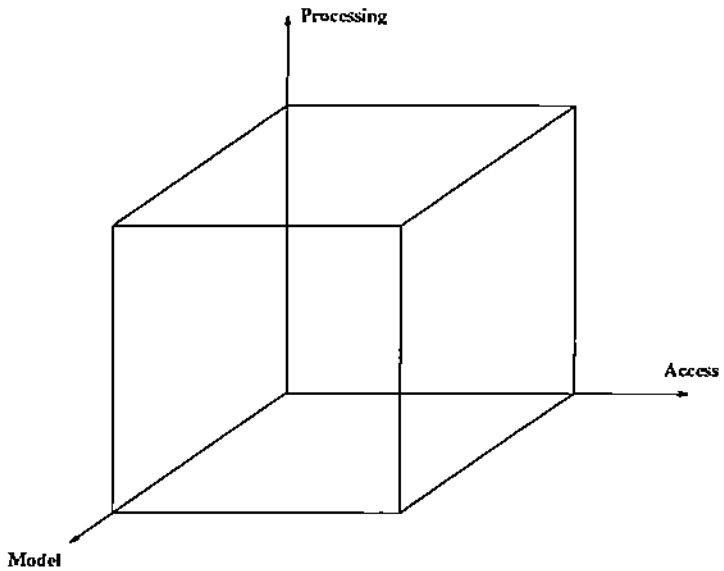

Figure 3: Dimensions of Database Heterogeneity

3.4 Dimensions of Database Integration

Three of the most widely used approaches to the creation of multidatabase systems are tightly-coupled federation, loosely-coupled federation, and interdependent data management. Tightly-coupled federations have schema integration, while loosely-coupled systems can define queries using multidatabase languages. Further, loosely-coupled systems do not maintain hard links into a mem-

ber databases. Only the databases which require integration are attached as needed to fulfill the transaction request. In interdependent data management, multidatabase interdependencies must be defined.

In these multidatabase approaches, database integration can be visualized in three dimensions:

- System integration: enables data to be accessed from more than one data base.
- Schema integration: provides a uniform global conceptual view of the multidatabase.
- Semantic integration: resolves data conflicts which might exist between component databases.

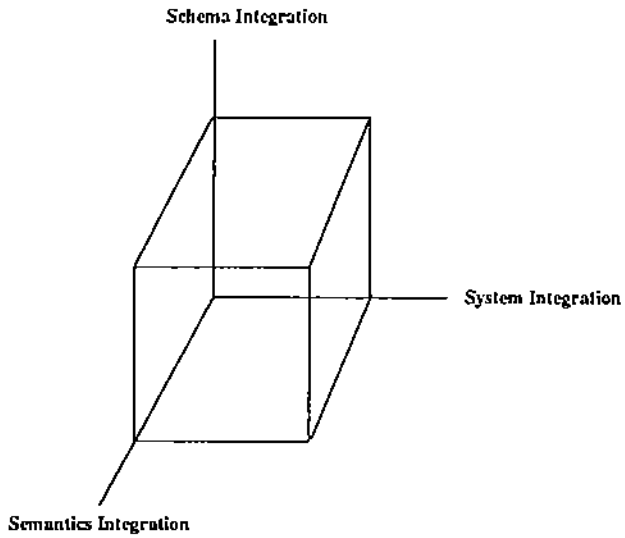

Figure 4: Dimensions of Database Integration

A database integration may be located at any point in the conceptual space defined in Figure 2, in that it can possess any of the dimensions of integration outlined above. These dimension will be discussed in further detail in the ensuring subsections.

3.4.1 System Integration

System integration provides both user transparency and system transparency. As multiple participating databases are usually distributed, the multidatabase must facilitate communications and allow users to access even the most remote database.

The design of transaction management for user transparency in multidatabase systems has been well studied, and it has been concluded that the preservation of the autonomy of participant database systems prohibits the utilization of the two-phase commit algorithm. A balance must be established between the demands of local autonomy and the prerequisites for full consistency and reliability. Many refinements have been suggested to better fit transaction management methods to particular multidatabase applications. In some instances, the basic requirements governing for database transactions can be relaxed.

3.4.2 Schema and Semantic Integration

A global multidatabase schema creates network and replication transparency by providing users with a uniform view of the federation of multiple databases. The common data model must be sufficiently inclusive to reflect all the features of the component systems. Currently, many researchers prefer the object-oriented data model as the canonical common data model for schema integration. More automatic tools are needed to reduce the expense of database integration.

Semantic data transparency forestalls data conflict and inconsistency among component databases. This is currently an area of active investigation, with many open questions yet to be resolved. As only semantically similar objects may be integrated, a first step must be to determine those objects with shared semantics.

Techniques for semantic reconciliation include [She91] :

- Comparison of objects. To determine similarity, the schema of objects may be compared with heuristic methods. For example, one may assume that the same name represents the same object, except when explicitly specified to the contrary.
- Formal/logic-based techniques. Users typically are required to specify semantic rules, such as the units of convergence, as part of the process of integration.
- Graphical facilities, CASE tools. Visual tools can aid users in specifying the semantic mapping between objects.
- AI/heuristic techniques. By applying methods such as expert systems, case-based reasoning,

and problem solving, computer systems can learn to recognize additional semantic cues. For example, a computer system can learn that some names are synonyms.

- Use of a large knowledge base. Some object semantics can be derived from a knowledge base.

System, schema and semantic integration must be tailored to suit a particular application and configuration of component databases. The extent of integration must also take into consideration, system performance, cost, and application goals.

4 Autonomy and its Effects

Autonomy is among the most significant properties of local (or component) database systems in a multidatabase system. Local autonomy arises from the originally independent design, implementation, and administration of those local systems. Informally, local autonomy in a multidatabase system is defined as the ability of each local database system to control access to its data by other database systems, as well as the ability to access and manipulate its own data independently of other systems. Local autonomy in multidatabase systems guarantees the independence of a local database system and that applications previously developed on that system continue to be executable, while ensuring the consistency and security of the local systems [DE89]. In this section, we explore the properties of local autonomy and its effect on various aspects of the multidatabase system.

The preservation of the local autonomy circumvents the costly and vastly cumbersome prospect of modifying local database systems. Local database systems have often been developed by independent companies, which may not wish to engage in such modifications. Local database systems of several years' standing often have hundreds of associated application programs, and this well-functioning complex must be preserved after integration into a multidatabase system. Any modification of these local database systems may create incompatibilities with some application programs and cause instability in the database systems, all undesirable consequences. Local database systems often belong to independent organizations which wish to retain a high degree of control over their database even after it is incorporated into a multidatabase system. For all these reasons, the

preservation of local autonomy is of great importance in multidatabase system integration.

Local autonomy has a significant impact on many aspects of multidatabase systems. Local autonomy permits easy addition to or removal of database systems from a multidatabase system. Furthermore, local autonomy renders the support of global applications in multidatabase systems much more difficult than in homogeneous database systems. The lack of understanding of the effects of local autonomy has greatly hindered the study and development of multidatabase systems. For example, the degree of compromise necessary are the part of local database systems to ensure the consistency of global databases is not fully understood.

4.1 Local Autonomy Requirements

Local autonomy, as mentioned above, defines the ability of each local database system to perform various operations on and to exercise control over its own data. For example, a local database system should be able to implement its own data model, catalogue management strategy, and naming conventions. It should also be able to exercise control over local transactions and global subtransactions (e.g., to delay or abort a transaction) in order to maintain the consistency of the local database. Local autonomy also defines the right of each local database system to make decisions regarding the service it provides to other local database systems. Local autonomy is necessary to guarantee that local users can continue to run local applications on their systems, regardless of integration, and to ensure that the basic consistency, security, and performance requirements of a local database system are met, while allowing other local database systems to access its data.

We may distinguish among four types of autonomy [DEK90, DELO89] :

Design autonomy refers to the ability of a local database system to choose its own design with respect to issues such as data model, query language, integrity constrains, and transaction processing strategy.

Execution autonomy refers to the ability of a local database system to decide whether and how to execute local operations without interference from external operations. For example, a local database system may assign a lower priority to external (global) operations than to local operations.

Communication autonomy refers to the ability of a local database system to decide whether,

when, and how to communicate with some or all other local database systems.

Association autonomy refers to the ability of a local database system to decide whether and how much to share its functionality and resources with others. For example, a local database system may allow external users to access only a portion of its data.

Design and association autonomy are static aspects of local autonomy; they define the right of a local database system to make decisions regarding static characteristics which are set at the time of integration. Execution and communication autonomy, on the other hand, are dynamic aspects of local autonomy which define the right of local database systems to make run-time decisions. To preserve design and association autonomy, the multidatabase system should not impose any restriction on local database systems (i.e., local database systems may use any data model and query language) and should also not require any modification to the local database systems. Similarly, preserving execution and communication autonomy implies that the multidatabase system exercises no control over local executions, other than the submission of global subtransactions.

It is usually difficult to effect a compromise regarding the requirements of design and association autonomy. For example, it is usually unacceptable to force a local database system to alter its data model or concurrency control protocol. On the other hand, the requirements of execution and communication autonomy may sometimes be compromised, it is usually acceptable, for example, for a local database system to abort a global subtransaction which the multidatabase system is prepared to commit.

The effects of design and association autonomy on multidatabase systems can be statically analyzed and therefore can be more easily comprehended. For example, the concurrency control protocol employed by each local database system is known at the time of integration, and based on this information, a global concurrency control protocol can be designed. In contrast, the effects of execution and communication autonomy, dependent as they are on the actual execution of applications, are much more difficult to observe and understand. For example, it is generally impossible for the multidatabase system to detect whether two global transactions indirectly conflict at a local site.

4.2 The Effects of Local Autonomy on the Global Concurrency Control

One of the main differences between homogeneous and heterogeneous database environments is the existence of local autonomies. In this section, we discuss the effect of these local autonomies on the design of global concurrency control algorithms for multidatabases. We also discuss difficulties of doing global concurrency control under the restrictions of these autonomies, and possible ways of remedying the problem.

Designing a concurrency control strategy for a heterogeneous database environment is different from designing a concurrency control algorithm for a homogeneous (distributed) database system and is much more difficult. The difficulties are primarily because a global concurrent controller must deal with heterogeneity and autonomy of underlying databases, in addition to the possibility that data may be distributed among various sites. In a homogeneous environment, there is only one concurrency controller to certify and produce the schedules. The concurrency controller has access to all internal information it needs to produce and/or certify the schedules. In addition , it normally has control over all transactions running in the system. The fact that concurrency control algorithms in traditional systems do not have to deal with the questions of autonomy and heterogeneity makes the problem sufficiently different that new algorithms must be designed for the heterogeneous environment. These difficulties manifest themselves even more gravely when addressing commitment protocols.

In a multidatabase system, local and global concurrency control must be addressed separately because of local autonomies. Local concurrency controllers guarantee the correctness (usually using serializability) of the executions of local transactions and global subtransactions at each local sites. The global concurrency controller, on the other hand, is responsible for retaining the consistency of the global database. Before the discussion of the difficulties of retaining the global database consistency, let us first introduce direct and indirect conflicts between operations.

In a concurrent execution, one operation might conflict with another operation in the sense that the effect of one influences the other. The influences can either be on the values read by an operation or on the current database state. Two conflicting operations are said to directly conflict if they both operate on the same data item, and indirectly conflict otherwise. Similarly, we say

that two global transactions directly conflict if they contain directly conflicting operations. If two global transactions are not directly conflicting but contain indirectly conflicting operations, then they indirectly conflict with each other.

To maintain global database consistency, a global concurrency controller has to be able to detect and resolve direct and indirect conflicts among global operations properly. Direct conflicts involve global operations only, and therefore can be easily predicted by a global concurrency controller. Indirect conflicts among global operations , however, might be introduced by local operations. Since a global concurrency controller has no direct control over local transactions and no direct access to information about local executions, it cannot detect indirect conflicts with certainty. This makes Global Concurrency Control difficult.

4.3 The Effects of Local Autonomy on the Global Transaction Manager

A global transaction manager coordinateds the execution of global transactions within a multidatabase system Ideally, the global transaction manager should enforce the ACID (atomicity, consistency, isolation, and duration) [HR83] properties on global transactions. This objective, however, is hampered by the constrains imposed by the autonomy of local database systems. While local database systems incorporate their own concurrency controllers to ensure that local transactions are run in a serializable fashion, improper synchronization can nevertheless occur at the global level.

Conflicts are even more likely to occur regarding the commitment of global transactions. To maintain local consistency, each local database system can unilaterally commit or abort a global subtransaction of a global transaction executed as a local transaction. This action may be taken without consulting the global transaction manager, possibly placing the global transaction in an unacceptable state. A local component system in a multidatabase sees itself as isolated, with no knowledge of the existence of other local systems and of the multidatabase system. For this reason, the atomicity of global transactions cannot be achieved if full local system autonomy must be retained.

4.4 The Effects of Local Autonomy on Schema Integration

Schema integration, which enables navigation and global data access in multidatabase systems, is another significant aspect of system design. Schema integration involves combining *component schemas* (the schemas of the local database systems) into an integrated global schema. Again, the constrains posed by the maintenance of local autonomy complicate the achievement of schema integration. These abstracts include [KS91]:

- **Data Model Heterogeneity.** Local database systems may employ different data models; for example, these may be relational, hierarchical, object-oriented, or entity-relationship.
- **Structural Heterogeneity.** Data may be structured differently in various local systems. In one local database system, "author" may be an attribute of a book entity, while another local database system may designate "author" as its own entity with an m:n relationship to book entities.
- **Unit Heterogeneity.** One local database system may store measurements in meters, while another may use feet.
- **Access Heterogeneity.** Local database systems may employ different language semantics for access data (eg. relational systems using SQL, text systems using text-enriched semantics)
- **Type Heterogeneity.** Two attributes that represent the same concept may be declared as different types. This may occur even when homogeneous systems are integrated, for example, one local database system may store social security numbers as a character string, while another stores them in integer form.
- **Name Heterogeneity.** Two local database systems may contain entities that are semantically equivalent but are assigned different names; e.g., "customer" and "client". In addition, attributes that are semantically equivalent may be named differently.
- **Naming Conflicts.** Two local database systems may each contain an entity with the same name but representing different objects.

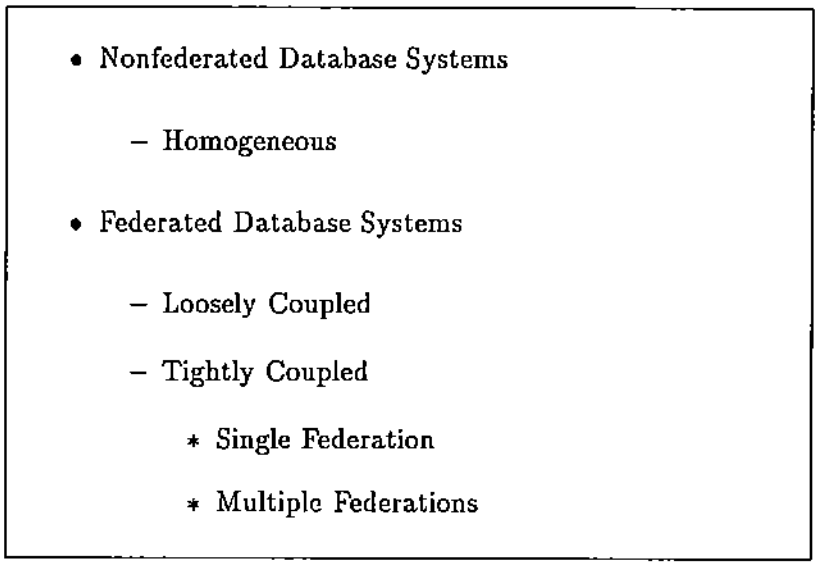

Figure 5: Multidatabase System Taxonomy

Schema integration is further complicated by the dynamic nature of component schemas. Under the terms of local autonomy, local database systems can freely modify their database schemas, disrupting the consistency between local schemas and the integrated global schema. Fortunately, controlled inconsistency, rather than strong consistency, is often sufficient for effective performance. For example, inconsistent data may be usable if it is at most one day old.

5 Architecture

5.1 Architectural Taxonomy

We shall now summarize and compare the various architectural models applied to multidatabase systems design. As stated earlier, a multidatabase system is a database system that supports the execution of operations on multiple component database systems. These component systems may be either centralized or distributed database systems. The basic multidatabase system taxonomy is shown in Figure 5.

This taxonomy involves the following classifications:

- **Nonfederated Database Systems.** A nonfederated multidatabase system is integrates component systems that are not autonomous but which may still be heterogeneous. In a nonfederated multidatabase system, no distinction is made between local and non-local users and between local and non-local transactions. As the rubric of multidatabase system is sometimes applied to those database systems with autonomous multiple component systems, such a definition would exclude nonfederated systems. The homogeneous taxonomy covers those systems that provide homogeneous access, execution and data models.

- **Federated Database Systems.** A federated multidatabase system integrates component systems that are autonomous. The federation therefore has limited control over these component systems and will have only partial knowledge of their actions. In federated multidatabase systems, there is a definite distinction made between local and non-local users. Local users interact directly with the component systems, and the federation has no direct knowledge of or control over the transactions they execute. Local users are restricted, however, in that the transactions they execute may only access data residing at the component database system with which they interface, Non-local users execute transactions through the federation, but these transactions may access multiple component database systems in the federation. There are two subvarieties of federated multidatabase systems [SL90]:

  - **Loosely Coupled.** In loosely coupled federated database systems, system users are largely responsible for the administration of the federated system. There is no central authority that controls the creation of or access to of data. Each component system is responsible for constructing global schema view and for processing queries that access remote component systems.

  - **Tightly Coupled.** Tightly coupled federated database systems include a central authority responsible for the administration of the federated database system. This central authority has control over global schema view(s) and global query processing.

    * **Single Federation.** A tightly coupled federated database system is considered to have a single federation if it supports only a single federated schema.

* **Multiple Federations.** A tightly coupled federated database system is considered to have multiple federations if it supports multiple federation schemas.

5.2 Architectural Comparison

Nonfederated vs. Federated. A nonfederated system facilitates support of certain important properties, such as concurrency control and atomic commitment. The traditional algorithms for these procedures (two-phase locking and two-phase commitment) assume that component systems can be modified as desired to implement the appropriate algorithms . While this can easily be accomplished in a nonfederated system, the constrains of local autonomy in integrated systems may interfere with the implementation of transaction management algorithms.

In many instances, however, a nonfederated approach is incompatible with the system context. In situations such as the following, a federated approach must be utilized:

- **Pre-existing Systems.** Within an organization, there may be pre-existing systems that are considered too costly to replace, yet a global database system may be desired. A multidatabase system spanning multiple organizations will also need to accommodate the features of its component database systems. In both these systems, a federated approach will be required.

- **Development Using Off-the-shelf Products.** Even given a centralized authority over an entire system, organization with no pre-existing systems, economic reasons may preclude building a system from the ground up, and off-the-shelf database systems may be used. Generally, the modifications that can be made to off-the-shelf products are very limited, as the source code is usually not provided.

Loosely Coupled vs. Tightly Coupled. The loosely coupled approach to federated database systems provides ensures a greater degree of autonomy for the component systems, since no central authority is imposed. In addition, loose coupling generally tends to scale better to very large systems, since it is difficult to support a central authority over such systems. On the other hand, the maintenance of consistency is more difficult in a system that lacks a central authority.

Single Federation vs. Multiple Federations. Multiple federations permit greater flexibility in viewing the data held in a tightly coupled federated database system. Multiple schemas can be created to tailor access to a variety of of interest and security considerations. The maintenance of consistency between multiple and local schemas and between various multiple schemes is, however, a challenging project. On the other hand, a single massive global schema will not scale well to a very large multidatabase system.

6 Concluding Remarks

Heterogeneous database systems are being developed in response to the need to bridge disparate sources of information which exist within many organizations today. This disparity is frequently caused by the acquisition and merger of organizations or changing business requirements that demand new applications or innovations that result in enhanced qualities of information.

Transparency as well as autonomy are key aspects for consideration in a multidatabase system. These aspects have significant impact on the complexities comprised in constructing an integrated global schema of the component database systems.

A newly commercial multidatabase system was released by Molecular Design, Ltd. in September 1991 [Sch93]. INtersect, embodied within the ISIS/Host product, was delivered to the pharmaceutical marketplace to provide transparent integration of heterogeneous database systems. It is an object-oriented multidatabase system that provides loosely coupled federations of heterogeneous databases. Using local schema's embodied in IIviews(Heterogeneous Views), it forms links dynamically when the user or application requests assembly of data within the schema.

Data model heterogeneity is promoted through a nested object model used for linking various database sources. This permits relational, hierarchical, CODASYL, object-oriented databases to maintain their data relationships in the integrated view that is provided to the application or end-user. These relationships are imported into the integrated view without requiring the reconstruction of those already exist.

INtersect supports access heterogeneity through the ability to support query operations and data

types that are unique to the respective heterogeneous sources. This allows users who are familiar with the strengths of the autonomous systems do not have to compromise in using the multidatabase system. A single query model for access (eg. SQL) was inadequate, as a requirement was to support the semantics of the respective homogeneous databases (eg. Text, Chemical Structure Search).

The object-oriented nature of INtersect has been carried through to the mechanism used to interface new database sources. Extensibility is promoted through the ability for gateway interfaces to maintain private methods for activation by the multidatabase system. These methods can extend to definitions for new datatypes, as well as their manipulation.

References

[BCD+93] Omran A. Bukhres, Jiansan Chen, Weimin Du, Ahmed K. Elmagarmid, and Robert Pezzoli. InterBase : An Execution Environment for Global Applications over Distributed, Heterogeneous, and Autonomous Software Systems. *IEEE Computer*, August 1993. (to appear).

[Dat77] C. J. Date. *An Introduction to Database Systems.* Addison Wesley, second edition, 1977.

[DE89] W. Du and A. Elmagarmid. Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 347–355, Amsterdam, The Netherlands, August 1989.

[DEK90] W. Du, A. Elmagarmid, and W. Kim. Effects of Local Autonomy on Heterogeneous Distributed Database Systems. Technical Report ACT-OODS-EI-059-90, MCC, February 1990.

[DELO89] W. Du, A. Elmagarmid, Y. Leu, and S. Ostermann. Effects of Autonomy on Global Concurrency Control in Heterogeneous Distributed Database Systems. In *Proceedings of the Second International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, pages 113–120, Gaithersburg, Maryland, October 1989.

[DW88] C. J. Date and C. White. *A Guide to DB2.* Addison Wesley, second edition, 1988.

[EGLT76] K. Eswaran, J. Gray, R. Lorie, and I. Traiger. The notions of consistency and predicate locks in a database system. *Communications of ACM*, 19(11):624–633, 1976.

[ELLR90] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A Multidatabase Transaction Model for InterBase. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 507–581, Brisbane, Australia, August 1990.

[GMK88] H. Garcia-Molina and B. Kogan. Node Autonomy in Distributed Systems. In *Proceedings of the First International Symposium on Databases for Parallel and Distributed Systems*, pages 158–166, Austin, Texas, USA, December 1988.

[HM85] D. Heimbigner and D. McLeod. A federated architecture for information management. *ACM transaction on office information systems*, 3(3), July 1985.

[HR83] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4), July 1983.

[KS91] Won Kim and Jungyun Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *Computer*, 24(12):12–18, December 1991.

[ME91] James G. Mullen and Ahmed K. Elmagarmid. On the impossibility of atomic commitment in multidatabase systems. Technical Report CSD-TR 91-029, Department of Computer Sciences, Purdue University, April 1991.

[ÖV91] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Inc., 1991.

[Sch93] T. Schaller. The INtersect Concept for Multidatabase System Integration in the Pharmaceutical Industry. *ACM SIGMOD Record*, 22(2), 1993.

[She91] A. Sheth. Semantic Issues in Multidatabase Systems - Preface by the Special Issue Editor. *SIGMOD Record*, December 1991.

[SL90] Amit P. Sheth and James A. Larson. Federated databases systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, pages 183–236, September 1990.

[Vei90] J. Veijalainen. *Transaction Concepts in Autonomous Database Environments.* R. Oldenbourg Verlag, Germany, 1990.