1993

A Taxonomic and Analytical Survey of Multidatabase Systems

Tony Schaller

Omran A. Bukhres

Jiansan Chen

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

Report Number:
93-040

A TAXONOMIC AND ANALYTICAL SURVEY OF MULTIDATABASE SYSTEMS

Tony Schaller
Omran A. Bukhres
Jiansan Chen
Ahmed K. Elmagarmid

CSD-TR-93-040
July 1993
(Revised November 1993)

A Taxonomic and Analytical Survey of Multidatabase Systems

Tony Schaller
Molecular Design Ltd.
2132 Farallon Drive
San Leandro, CA 94577
tonys@molecular.com

Omran A. Bukhres, Jiansan Chen, and Ahmed K. Elmagarmid
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{bukhres, jchen, ake}@cs.purdue.edu

Contents

1 Introduction

The successful integration of computational process and data from disparate sources is a complex process. An integrated system interconnects pre-existing systems to support global applications accessing more than one component system. For many global applications which require simultaneous access to multiple systems, such an integration presents an attractive alternative to a single system and offers enhanced performance and flexibility. Unlike traditional distributed systems, an integrated system interconnects component systems in a bottom-up fashion, thereby allowing existing applications developed on the component systems to remain executable without modification.

A multidatabase system is such an integrated system, in which heterogeneous and autonomous component database systems are unified on a global level so as to support global applications accessing more than one component database. A multidatabase interoperation provides an integrated view of the data and resources of these applications and presents global users with transparent access to the information stored in component databases, without violating the local autonomy or independent administration of these component databases. The multidatabase architecture is typified by the French Teletel system which allows 1.8 million users to access over 1000 separate databases. Modern information-dependent technologies requires that pre-existing information sources and systems be interconnected to allow users to access these resources in a unified format.

As the number of multidatabase prototype and commercial systems expands, the need for a comprehensive research analysis of these systems becomes more pressing. Such an analysis would examine the nature of multidatabase systems, summerize the findings gleaned from current developments, and provide suggestions for the future research in this area. Researchers involved with the development of multidatabase systems could draw upon this analysis for the guidelines when formulating new theoretical models and practical products, while end-users could employ these results as standards in assessing the required functions provided by different multidatabase products.

Several investigations along this line have already appeared in literature. In [SL90], multidatabase systems are classified as non-federated and federated. A non-federated database system is an integration of non-autonomous component database, presenting to its users the appearance of a distributed database system. A federated database system consists of component database systems that, while still autonomous, nonetheless participate in a federation to allow partial and controlled data-sharing. A federated database system can be loosely coupled or tightly coupled. A tightly coupled federated database system has at least one global schema which is maintained by the federation, while a loosely coupled federated database system has no global schema.

In [TTC+90], the capabilities of multidatabase systems are described as including schema integration, distributed query processing, distributed transaction management, administrative functions, and accommodating different types of heterogeneity. Several multidatabase systems, including Ingres and Sybase, are analyzed in accordance with the listed capabilities.

In [BHP92], multidatabase systems are classified along a continuum from tight to loose coupling, in the following categories: distributed databases, global schema multidatabases, federated databases, multidatabase language systems, homogeneous multidatabase language systems, and interoperable systems. A distributed database system is equated with a non-federated database system as classified by [SL90], while a global schema multidatabase system is tightly coupled with a single global schema, and a federated database system is tightly coupled with several global schemas. A multidatabase language system provides no global schema, leaving its users to rely on query language tools to access component systems. A homogeneous multidatabase language system is a special multidatabase system with homogeneous component systems and an interoperable system acts as a front-end to component systems.

Although the above-mentioned analysis of multidatabase systems has provided much useful information, a more in-depth investigation which would offer more comprehensive insights has yet to be attempted. The present research attempts to redress that lack.

In this paper, we first construct a logical architecture of multidatabase systems and explore the major issues related to multidatabase integration. These tools are then applied to the analysis of some existing multidatabase systems. The features of these systems are then summarized according to the following criteria: 1) integration of disparate systems; 2) provision of schema translation and schema integration; 3) support of various local data model; 4) transactions support; and 5) maintenance of autonomy of component systems.

2 Major Issues of Multidatabase Integration

The principle obstacle to multidatabase integration lies in the autonomy and heterogeneity of component systems. The autonomy of component systems, which renders the multidatabase system unable to modify, control, and closely monitor these component systems, may arise from a variety of causes. The owner of a component system may be unequipped to make modifications. Also, the component system may belong to a different organization than the one developing the multidatabase system. Furthermore, alterations to well-established software systems would create incompatibilities with the many application programs which they service. The retention of local autonomy is therefore of utmost importance in multidatabase system integration. The second characteristic of component systems, heterogeneity, may be manifested in a variety of aspects, including query language, data model, concurrency control, and atomicity control.

The heterogeneous and autonomous nature of component database systems greatly complicates the theoretical foundation of a multidatabase system. For example, consider a multidatabase transaction that transfer data from one component database system to another. Because these component systems are autonomous, they can unilaterally commit or abort their section of the operation after it is completed or failed. In this instance, if one of these two component database systems has committed while the other one has aborted, the data transferred may be either lost or duplicated. Many approaches have been proposed in the literature to address such problems, which arise from the heterogeneous and autonomous nature of the component database systems in a multidatabase system.

In general, multidatabase integration occurs on four levels, the **user interface level**, the **schema level**, the **transaction level**, and the **interface level**. The **user interface** provides users with a global query interface employing a unified format. It may also provide a programming interface which allows users to embed multidatabase operations into other programming languages, permitting more flexible applications. The **schema integration** of a multidatabase system provides global schemas and views into the schemas of underlying component database systems. This level serves to mask the details of data stored in component database systems and to provide a uniform view of these data. The **transaction level** of a multidatabase system builds upon the transaction management facilities of component database systems to support transaction applications. Finally, the **interface level** transforms command text and data between the multidatabase format and that of individual component database systems. The four levels of multidatabase integration will now be discussed in further detail.

2.1 System Architecture

A logical multidatabase architecture is graphically presented in Figure 1, which clarifies the relationship among the four levels of multidatabase integration. In this figure, $Query_i$ indicates a query originated by a user of the multidatabase system, while G_i is a global transaction which is the consistent and reliable execution of $Query_i$. G_i accesses Component Database System$_1$ through Component Database System$_n$. $G_{i,m}$ $(1 \leq m \leq n)$ indicates a global subtransaction which consists of all the operations of G_i on Component Database System$_m$. $L_{i,m}$ $(1 \leq m \leq n)$ indicates a local transaction on Component Database System$_m$ which is transformed from $G_{i,m}$. $L_{j,1}$ and $L_{k,n}$ indicate local transactions submitted directly from the user to Component Database System$_1$ and Component Database System$_n$, respectively.

A user can initiate a query through the global user interface provided by a multidatabase system. After decomposition and translation with the aids of the global schema, the query is submitted to the Global Transaction Manager (GTM) as a global transaction (G_i). The GTM then surveys the dependencies among the subtransactions of G_i ($G_{i,1}$, $G_{i,2}$, ..., and $G_{i,n}$) as the basis for selecting one to be submitted for execution first. Without loss of generality, let us assume ($G_{i,m}$, $m \leq n$) is submitted first by the GTM. $G_{i,m}$ is received by an interface superimposed on Component Database

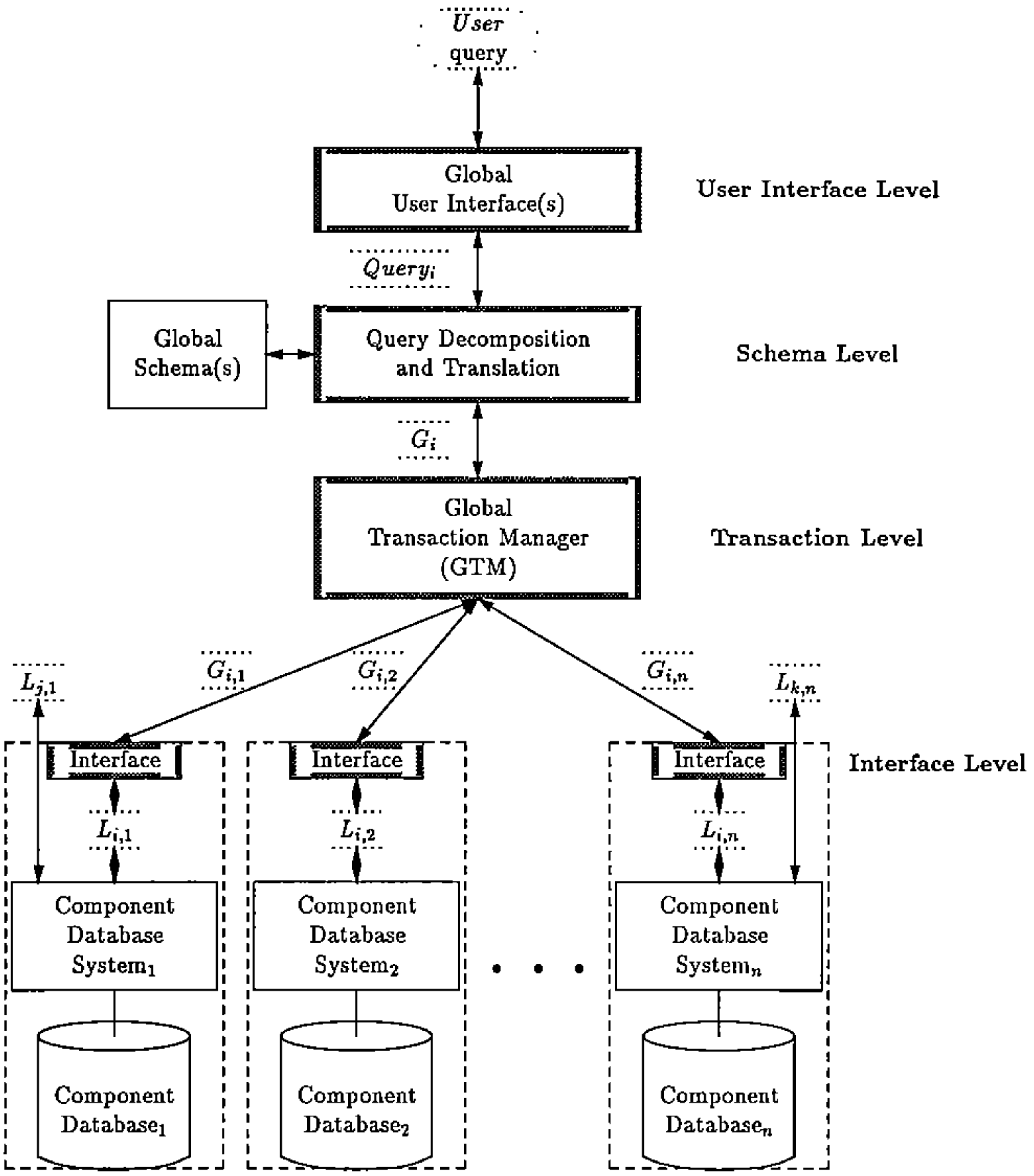

Figure 1: A logical multidatabase architecture

System$_m$. This interface, which is a special designed agent of the multidatabase system, performs all necessary format transformations and sends the transformed $G_{i,m}$ as a local transaction $L_{i,m}$ to Component Database System$_m$. At the same time, Component Database System$_m$ can also accept local transactions from users directly. The result of $L_{i,m}$ is sent to the interface, which translates the result into a uniform format defined by the multidatabase system. The results are then sent to the GTM as the result of $G_{i,m}$. The GTM continues to execute other subtransactions of G_i until it is able to decide and act to commit or abort G_i, the final result of which is sent to the user.

This logical architecture can be readily tailored to meet the requirements of various multidatabase systems. For example, if each multidatabase interface is embedded in its associated component database system, the multidatabase system can then be seen as a non-federated or a distributed database system, as defined by [SL90] or [BHP92], respectively. If only multidatabase interfaces are provided for each component database system, the multidatabase system can be categorized as interoperable. If all component database systems are identical, the multidatabase system is simply a homogeneous multidatabase system. If the global user interface provided is simply a database query tool, the multidatabase system is a multidatabase language system.

The above discussion pertains to the logical architecture of the system. The physical design of a multidatabase system falls along a continuum defined by two extremes. At one extreme is a centralized system in which all global queries are sent to a central node, where they are processed, executed as global transactions, and their subtransactions submitted to appropriate component database systems for execution. While such system is easy to design and implement, its central node becomes a hot-spot, the failure of which immobilizes the entire system. At the other extreme is a totally decentralized system in which each node can independently process global queries originated at that location. As all the nodes work independently, the failure of a single node is not propagated to other nodes. The successful design of such a decentralized architecture has become a central aim of multidatabase system designers.

2.2 Schema Integration

Much of the early investigation into multidatabase integration concentrated on global schema design, and in particular, on schema translation. A good review of some of the early work on schema translation can be found in [HF83, Lar83], while the entire topic of schema integration is covered in [BLN86, SL90]. Many different approaches for schema integration have been discussed, and some are presented in [BL84, DH84, NSE84, NEL86, SL88, LNE89, SG89].

Schema translation involves translating the schemas of component systems into a common (or canonical) data model format. The output of this process will typically consist of common data model schemas and data mapping and query translation rules. Schema integration addresses the integration of multiple component system schemas into unified global (or partial global) schemas. Figure 2 provides a pictorial representation of these two integration phases. Several variations of the figure shown are possible:

- No integrated views are provided. The purpose of the database then is to provide a common query language. Navigational queries can be performed, or keywords and partial matching can be used in queries to attempt to retrieve the desired data.
- A single integrated view is provided that represents the entire global database status.
- Each component system has its own integrated view, as described in [HM85].
- Multiple local views or multiple views of integrated schemas are supported.
- Multiple global query processors (or other applications) access the integrated views. In the federated approach, each component system is in effect a global query processor.

While schema translation to a common data model overcomes the heterogeneity of data models, many other obstacles to schema integration still remain. If the objects in the different database are all semantically unrelated, there may be a:

- **Naming Conflict**; two databases may each have classes bearing the same name but representing different concepts.

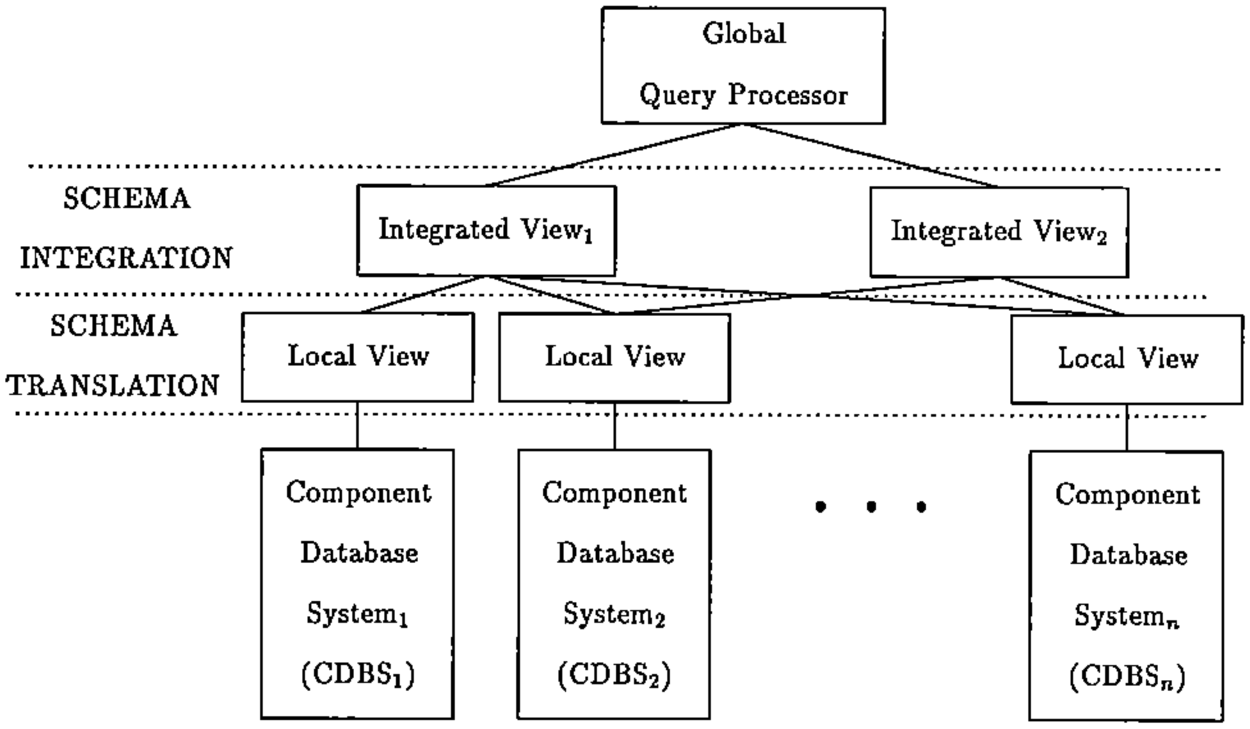

Figure 2: Multidatabase Integration

Such conflicts can be resolved by simply renaming one of the classes in the integrated view. However, if some of the objects are semantically related, many conflicts can occur, including:

- **Structural Conflict;** data may be structured differently in each component system. Objects in different component systems that represent the same semantic concept may have a different number of subobjects or may include subobjects with no semantically corresponding subobject in the system.

- **Unit Conflict;** objects may use different units. For example, two databases might include "car" objects with an attribute called "length" of (primitive) type float, but one database may store lengths in feet and the other in meters.

- **Type Conflict;** two attributes that represent the same concept in related objects may be declared as different types. For example, one component system may store a social security number as a character string in an "employee" object, while another stores it as an integer.

- **Name Conflict;** two database systems may have classes that are semantically equivalent but use different names, such as "customer" and "client." In addition, attributes that are semantically equivalent may be named differently.

A more detailed discussion of data and schema conflicts that can occur in multidatabase systems is presented in [KS91] and [BLN86].

2.3 Multidatabase Transaction Management

Ideally, multidatabase transaction management should enforce the ACID properties (atomicity, consistency, isolation, and duration) on multidatabase transactions (termed global transactions). However, because data in a multidatabase system are maintained by component systems, multidatabase transaction management is only responsible for controlling global concurrency and global atomicity and recovery.

Global concurrency control is necessary even though all component systems maintain serializability [DELO89, ÖV91]. Concurrency control improves the response times of database queries and prevents longer transactions from delaying shorter transactions. Traditionally, serializability is used as the correctness criterion for concurrency control; it requires that a concurrent execution of a set of transactions be equivalent to a serial execution of these transactions.

Global atomicity and recovery control is a prerequisite for the enforcement of the atomicity of global transactions in a multidatabase system. Even though all component systems support atomic transactions, an atomic commit protocol is still required. For example, consider a global transaction G which submits a subtransaction G_1 at component database C_1 and a subtransaction G_2 at component database C_2. If both G_1 and G_2 are committed or aborted, G can be said to be atomic. However, if one subtransaction is committed while the other one is aborted, G is terminated in an unacceptable state.

The problems involved in multidatabase transaction management were first raised in [MB81], and a more detailed discussion was represented in [GPZ86]. In the following years, multidatabase transaction management has been the subject of extensive investigation. Examples of proposed algorithms growing out of these research efforts include [AGMS87, BS88, Pu88, KLS90, BST90, DEK91, GRS91, MRB+92, ZE93]. In [BGMS92], an overview of multidatabase transaction management is presented.

Traditional transaction management strategies, which presume full knowledge and control by the transaction manager over the managed transactions, are best suited to database systems built from scratch. The assumptions upon which they are based do not extend to multidatabase environments, which integrate existing systems in which transactions are scheduled independently by the autonomous transaction managers of component systems. Global serializability and global atomicity and recovery have therefore become central to the successful development of multidatabase transaction management schemas.

2.3.1 Global serializability

An execution is serializable if it is semantically equivalent to a serial execution in which transactions are executed sequentially. Global serializability implies that all local executions on component database systems are serializable and that the serialization orders of transactions at different component database systems are consistent. These conditions are not readily met, as the serialization orders are generally not known to the multidatabase transaction manager. Serialization orders may also differ from the ordering of physical events, such as the beginning or end of an execution (submission or commitment), which are controllable at the multidatabase level. Two approaches, among many proposed algorithms, appear most promising to address this problem.

The first approach may be applied in those multidatabase environments in which the serialization order is consistent with some physical event order. For example, if the transaction manager of each component database system is *rigorous* [BGRS91], in that it does not begin the execution of a transaction until all previous conflicting transactions have committed, then the serialization order will be consistent with the commitment order.

An alternative approach would involve first relaxing the requirement of global serializability and then resolving any inconsistencies by controlling other (non-scheduling) aspects of execution.

Typical of this method is the quasi-serializability (QSR) approach [DE89]. Instead of coordinating serialization orders at different component database systems, QSR ensures only a consistent execution order. That is, if two transactions access more than one component system, they are executed sequentially in a consistent order at all component systems. Possible inconsistencies due to non-serializable executions are prevented by controlling the flow of information among global subtransactions.

The first approach offered above is particularly applicable to the integration of database systems, as most existing commercial database systems are rigorous. With problems of more general integration, where not all component systems are rigorous, the second approach would be appropriate.

If the serialization orders of component systems can be obtained by the multidatabase transaction manager, global serializability can be easily achieved. This situation forms the basis of several optimistic algorithms, such as the SuperDatabases approach [Pu88]. When a global subtransaction is completed, the corresponding component system reports the local serialization order to the multidatabase transaction manager. The multidatabase transaction manager compares the serialization orders of each committing global transaction with those of all other recently committed transactions, and issues a verification if the orders are compatible at all component systems.

2.3.2 Global atomicity and recovery

A greater challenge is presented by the requirement, arising from the properties of atomicity and durability of global transactions, that all the subtransactions of a global transaction either commit or abort. The traditional approach (two-phase-commit) relies on the prepare-to-commit state provided by all component systems to prevent unilateral commit/abort and to survive failures. Such an approach may not be feasible when integrating existing systems that do not all support a prepare-to-commit state.

One solution to this difficulty would be to apply a transaction model, such as Flex Transactions [ELLR90], which permits a more flexible commitment protocol. It is also possible to redo or retry unilaterally aborted subtransactions or to semantically undo (compensate) unilaterally committed subtransactions [Gra78, GM83, KLS90, BGMS92].

In summary, despite a flurry of research activity on the subject, the maintenance of global consistency within a multidatabase and of the atomicity of global transactions are still challenging problems to which no satisfactory solution has yet been found [CAC91]. Furthermore, most investigations have concentrated on the theoretical aspect of these problems, very little attention to practical concerns.

2.4 Global User Interfaces and Multidatabase Interfaces

Global user interfaces permit access to the multidatabase system as a whole. Unlike the problems of schema integration and transaction management, very little research has been specifically targeted toward global user interfaces, although they often constitute a sidebar to a discussion of schema integration and the description of prototype multidatabase systems. This may be attributable to the highly practical nature of the question of user interfaces, rendering it a time-consuming and potential underworking area for exploration.

Traditionally, there have been three levels of user interfaces for database systems. User interfaces designed for specific application environments usually take the form of fill-in tables [Row85] or pull-down menu [KM89]. Users of such user interfaces must follow a pre-set format, with guidance provided by the interface itself. While user interfaces of this type are easy to use, they are fixed for specific applications and therefore inflexible. A second variety of user interfaces, typified by SQL (Structured Query Language), is designed for non-experts. Users can make *ad hoc* queries to database systems, where they are processed, run against the data, and the desired results returned. The most sophisticated user interfaces are interface primitives that can be embedded in a programming language such as C or COBOL. User interfaces of this sort provide a flexible and efficient, although more demanding, methods to use the data stored in a database. In fact, most fill-in table format user interfaces are built on such sophisticated user interfaces.

At present, most existing multidatabase systems employ SQL or similar interfaces. because SQL is popular among traditional database users and all commercial database systems are equipped with an SQL user interface.

Multidatabase interfaces perform the mapping between a multidatabase and its component systems. This area is also frequently addresses along with the issues of schema integration and transaction management. Like the question of global user interfaces, the problem of multidatabase interfaces is also far more a practical than a theoretical issue.

Many of the functions of schema integration and transaction management can be performed through multidatabase interfaces. The translation of local schema of a component system into the global schema format specified by the multidatabase system can be performed by each multidatabase interface. The complexity of a multidatabase interface is dependent largely on the component system integrated. If all the component systems are relational databases, the commonality of their features simplifies the construction of these interfaces. On the other hand, if the component systems are heterogeneous, involving a mixture of both database and various non-database systems, the resulting heterogeneous interfaces are more difficult to construct.

Not only the nature of component systems themselves but also the standard user interfaces provided by these component systems play an important role in the development of multidatabase interfaces. To preserve local autonomy, a multidatabase interface must be tailored specifically for compatibility with the standard interfaces provided by its associated component system. For example, some systems, such as the Sybase and Ingres DBMSs, provide both a query (command) and a programming interface; other systems, such as the UNIX shell, provide only a command interface. The nature of the multidatabase interface to be developed is shaped by the particular component system interface involved. The development of multidatabase interface to accommodate query (command) interfaces brings with it the following problems:

- because component systems tend to output a mixture of useful and auxiliary data, output interpretation is complex;

- the multidatabase interface may be unable to properly interpret error messages sent to an error file by a component system; and
- different versions of a component system may use different output and error message formats. These difficulties can be bypassed by formulating a multidatabase interface to suit the programming interface of its component system whenever possible.

Furthermore, the programming interface of a component usually provides more functions than its query (or command) interface, allowing the multidatabase interface to support more applications.

A global user interface and a multidatabase interface for each component system may be considered to be the minimum requirement of a multidatabase system.

3 Taxonomy

The following five taxonomic categories form the basis of our review of the characteristics of multidatabase systems:

- **Integration of Disparate Systems.** A multidatabase system may include only databases as its component systems, thus severely restricting the scope of multidatabase integration. Since many non-traditional distributed transaction applications demand the integration of non-database systems such as CAD/CAM packages, multidatabase systems are now frequently required to incorporate various non-database systems.
- **Provision of Schema Translation and Schema Integration.** Schema translation describes the process of translating the heterogeneous data models of component systems into a common data model, such as an object-oriented model. Schema integration, as described in Section 2.2, addresses semantic issues that are not encompassed in the process of schema translation. The provision of schema integration reinforces the tightly coupled architecture of the multidatabase system.

- **Support of Various Local Data Models.** Many different data models may be employed by component systems. For example, a component database system may employ either a relational, network, hierarchical, entity-relationship, or object-oriented data model, while a non-database component system may use any format to store and process data. A flexible multidatabase system must support a variety of local data models.

- **Transaction Support.** A global transaction manager for a multidatabase system enforces the ACID properties (atomicity, consistency, isolation, and duration) on global applications over the system. For many applications, it must be at least semantically guaranteed that either all or none the operations of these applications will execute. For example, a customer who requests a flight ticket and a hotel room would like either both or none of the requests to go through.

- **Maintenance of Autonomy of Component Systems.** A multidatabase system should require no modification of its component systems. The owner of a component system may be unequipped to make such modifications. Moreover, the original developer of the component system may not be involved in the development of the multidatabase system. The retention of local autonomy is therefore of utmost importance in multidatabase system integration.

4 A Review of Current Research Prototypes and Commercial Systems

In this section, we review many of the multidatabase projects currently reported in the literature. These projects originate in a variety of countries and institutions. Some are academic and industrial prototype which serve as research vehicles to study specific problem areas, while others are fully commercial systems. The range of organizations involved and the number of projects reported are a testimony to the importance of this field, and both academic and industrial research institutions have invested heavily in multidatabase development. The reported functional content of each project is based on published reference materials.

4.1 A Review of Academic Research Prototypes

In this subsection, we discuss various multidatabase research prototypes developed in academic institutions. These prototypes are mainly research vehicles to study specific problem areas and usually provide only minimal system capabilities, including relatively user-unfriendly interfaces, simple security control, and incomplete error handling.

4.1.1 IMDAS

IMDAS is a tightly coupled federated system [KSL$^+$87] developed by National Institute of Standards and Technology and University of Florida; it has a single schema. The integrating data model is a semantic network data model capable of representing the complex structures and relationships and many integrity constraints found in a manufacturing enterprise. A fragmentation schema maps the global schema model to the underlying databases, supporting both horizontal and vertical partitioning of a given object class.

Existing database systems are front-ended by IMDAS modules supporting an internal query interchange form, which is an extended algebra of generalized relations corresponding to the modeled object classes, and a corresponding data interchange form, expressed in Abstract Syntax Notation 1 of ISO Standard 8824 and 8825. This common interface is readily mapped onto underlying relational and navigational databases. A library of supporting routines minimizes the effort involved in integrating new data systems and databases.

The user program phrases queries in an SQL-like language adapted to the model. The query is passed to the IMDAS in string form, rather than precompiled, to permit access by controllers programmed in non-standard languages. This mechanism can support an interactive interface, although none has yet been built. IMDAS supports both distributed updates (transaction management) and distributed retrievals (query management). The fragmentation schema does not currently support replication, however, which forms a significant limitation upon the system.

4.1.2 InterBase

The InterBase System is a multidatabase prototype developed at Purdue University [BCD+93] which is designed to provide a tool-based interface that facilitates application development in a distributed environment of heterogeneous software resources such as databases, tool libraries, and application programs. InterBase has been designed to manage the details of locating and starting remote services, transferring and transforming data among different services, managing failures, and controlling parallelism between multiple global applications running concurrently.

InterBase provides two different levels of user interfaces. A SQL interface, currently undergoing development, forms the higher level, permitting users to input *ad hoc* queries to InterBase. The InterBase Parallel Language (IPL) [CBE93], at the lower level, supports application program integration and distributed transaction processing and acts as an application programmer interface to InterBase. In this language, subtransactions of global transactions are executed in parallel whenever possible. IPL allows system programmers to specify all actions associated with a global transaction, such as control flow and data flow among subtransactions. A graphical interface is also provided over InterBase which aids users in writing and executing IPL programs.

InterBase employs Remote System Interfaces (RSIs) superimposed on each component systems to deal with the problem of heterogeneity. The RSIs are responsible for transforming commands and data between global transactions and component systems. RSIs therefore form a "homogeneous" interface toward these heterogeneous component systems, greatly reducing the complexity of the specification and execution of global transactions, while preserving the autonomy of these component systems.

One of the strengths of this architecture is its decentralized nature, The DFTM (Distributed Flex Transaction Manager) is distributed over all the machines from which global transactions are executed. As a result, a DFTM replica, which is invoked from the machine where a global transaction is generated, is responsible for the consistent and reliable execution of its associated global transaction. Exchange of information within Interbase is performed via computer network, so that each module of InterBase has location transparency. Each global transaction consists of subtransactions, each of which must be executed on a component system through its associated RSI. As the first step of its execution, a global transaction G_i must communicate with the relevant RSIs to arrange a consistent execution order of its subtransactions relative to those of other global transactions on these RSIs. In this way, correct execution of global transactions is guaranteed. The RSIs then execute the subtransactions of G_i in the specified order.

Distributed updates in InterBase are allowed as long as the global transactions are specified using Flex Transaction Models [ELLR90].

4.1.3 NDMS

NDMS (Network Data Management System) is a system developed at CRAI, Italy [ea85]. It includes support for IDMS, ADABAS, and RODAN database management systems hosted on IBM hardware and Ingres on DEC VAX hardware.

NDMS uses a relational data model as its global data manager. It involves three levels of abstraction: the NDMS internal schema, the application schema (conceptual schema), and end-user views (external schema). The NDMS internal schema is comprised of base relations which are defined as aggregations over the local database schema. The definition of these base relations require a data mapping to be specified for each local database, depending on its DBMS data model.

The interface of a local DBMS is made up of NDMS control software and a System Encyclopedia. This System Encyclopedia contains all information pertaining to a particular component system, including user, database mapping, and transaction definitions, as well as the complete NDMS internal schema definition.

A node data administrator is responsible for NDMS applications at each component system. Using SEQUEL view definition mechanism, it defines relational views as a collection of data abstractions (aggregations and generalizations) over the NDMS internal schema. The NDMS version of SEQUEL includes modifications to handle generalized abstractions. Predefined relational views are available to users for characterizing their specific data abstractions.

4.1.4 MRDSM

MRDSM, developed at INRIA (France) [Lit85], extends the MRDS relational database management system of HONEYWELL to support multiple databases. The goal of MRDSM is to accommodate semantic heterogeneity and thus provide uniform access to these databases. MRDSM operates on a specialized domain of multiple MRDS relational database management systems running on a HONEYWELL system. The query language is MDSL (similar to SQL), which is also the data manipulation language for MRDS. A global schema does not exist in MRDSM. Instead, users can create a conceptual schema, known as a multischema, with elements from local database schemas. Multischema is also associated with one or more dependency schemas which provide details such as inter-database dependencies, which include manipulation, privacy, and equivalence dependencies. Equivalence dependencies handle data incompatabilities and semantic mismatches.

After the removal of interdatabase dependencies that cannot be processed locally, a query on the multischema is decomposed into queries on local databases. A working database schema is then created to collect data from different databases, using an optimized collection strategy, and queries are then generated on these working classes. Finally, all data streams are combined and dependencies resolved.

MRDSM operates within a specialized domain, rather than a 'true' heterogeneous environment. Heterogeneity is dealt with at the semantic level by providing uniform access to all the databases encompassed under a single DMBS.

4.1.5 OMNIBASE

OMNIBASE is a multidatabase prototype developed at the University of Houston [REC+89]. It consists of a Global User Interface which utilizes a knowledge base to analyze queries sent by users and when necessary, remove ambiguous references. The analyzed user query is then sent to a Global Query Parser and Decomposer which decomposes the query into a set of subqueries. Next, a query evaluation plan is constructed, in the form of a program expressed in a Distributed Operation Language (DOL). The use of DOL allows the specification of complex multi-site data processing requests that may involve not only multiple databases but also other software packages. The query evaluation plan is subsequently executed under the control of the Query Evaluation Supervisor (QES), which directs the subqueries to their respective local systems. For each subquery, a Local Access Manager (LAM) translates the query into the local database language and submits it to the Local Database Management System (LDBS). An intermediate result of a local query produced under the control of a LAM may be then directed to some other site. LAMs are used to protect the local autonomy of the LDBSs participating in the multidatabase system and act as local agents of the MDBS. The results of subqueries are then combined to produce the final answer to the global query.

Because of the lack of centralized control in the design of LDBSs, different databases may use dissimilar data types to define data objects that logically belong to the same domain. Furthermore, data representation and precision may vary even for data of the same type. To accommodate this heterogeneity and possible inconsistency among databases, a multidatabase dictionary service must be provided either as a part of the MDBS or as an independent directory server. Whenever a new LDBS joins the MDBS or a member database is modified in a manner affecting its export schema, the appropriate schema information must be made available to the MDBS. The Knowledge Base serves this purpose.

4.1.6 PRECI*

PRECI* is a prototype of a generalized distributed database system developed at the University of Keele and the University of Aberdeen in collaboration with a number of research centers, mainly in Britain [DAODT85]. It is a generalized distributed heterogeneous DBMS with retrieval and limited update facilities. The local schema of each existing database is redefined, and each is then referred to as a node, with access to other databases, or nodes, accomplished via a relational algebraic interface.

The designers of PRECI* refer to their global data model as a canonical data model. It uses extended ANSI/SPARC architecture, and its conceptual schema (or canonical schema) is written in a relational form. The principal data manipulation language is the PRECI Algebraic Language (PAL), which offers such commands as Alteration, Rename, and Change Scale for data integration.

Each nodal database in PRECI* is fully autonomous, with its own nodal DBMS (NDMS) and nodal external schema (NES). The latter provides a PAL interface to the distributed database which uses PAL as the standard language for communications.

PRECI* permits a large number of local database management systems to participate as nodes, either as inner or as outer nodes. The inner nodes contribute to the global conceptual schema definition. If the number of participating nodes is large, some are designed as outer nodes, which do not contribute to the global database schema (GDS). The users at these nodes can achieve a partially integrated view by defining their own mappings. Queries from users at the outer nodes are dealt with in a similar manner to those from inner-node users. This strategy reduces the overhead involved in creating a GDS and GES (global external schema) for a large number of nodes.

The local database schema must be redefined to support a relational algebra or PAL. PRECI* allows global updates to be made to base relations only; global update requirements are submitted to local database management systems on an individual database basis. If the data are replicated, an update is performed only on the original copy and the result is propagated to other copies.

4.2 A Review of Industrial Prototypes

In this subsection, we discuss various multidatabase research prototypes developed in industrial research institutions. Like their counterparts in academic institutions, these prototypes are also mainly research vehicles to study specific problem areas and usually provide minimal system capabilities, including relatively user-unfriendly interfaces and simple error handling.

4.2.1 ADDS

The Amoco Distributed Database System (ADDS) [BT85] provides uniform access to preexisting heterogeneous distributed databases. The ADDS system is based on the relational data model and uses an extended relational algebra query language, as well as supporting a subset of the

ANSI SQL language. ADDS supports multiple federated schemas. Local database schemas are mapped into multiple federated database schemas, called Composite Database (CDB) definitions. These mappings are stored in the ADDS data dictionary, which is fully replicated at all ADDS sites to expedite query processing. A CDB is usually defined for each application, but multiple applications and users may share CDB definitions. Users must be authorized to access specific CDBs and relational views that are defined against the CDBs.

The CDBs support the integration of hierarchical, relational, and network data models. Local DBMSs currently supported include IMS, SQL/DS, DB2, RIM, INGRES, and FOCUS. Semantically equivalent data items from different local databases, as well as appropriate data conversion for these data items, may be defined.

The ADDS system includes geographically distributed mainframes running the VM and MVS operating systems and Sun and Apollo workstations running the UNIX operating system. A uniform network interface to these systems is therefore of great importance, and ADDS provides the Network Interface Facility for this purpose.

ADDs maintains the autonomy of its local database systems and does not require any modifications to local DBMS software. The only communication between ADDS and the local DBMSs is in the form of query submission and data retrieval.

4.2.2 DATAPLEX

DATAPLEX is a heterogeneous distributed database management system developed at General Motors Corporation [Chu90]. It allows queries and transactions to retrieve and update distributed data managed by diverse data systems in such a manner that the location of data is transparent to requesters. In this environment, different data management systems can run on several operating systems that may be connected by a variety of communication protocols.

DATAPLEX employs a relational model as its global data model. Since the data models found in dissimilar database systems structure data differently, the data definition for each sharable database in the heterogeneous distributed database system is transformed into an equivalent relational data definition or conceptual schema. The conceptual schema is implemented as a set of overlapping relational schemas, one for each location. The relations at each location represent data objects to be accessed by users at that site. Consequently, conceptual schemas are neither centralized nor replicated.

A prototype DATAPLEX system incorporates an IMS hierarchical DBMS running under the MVS operating system and an Ingres relational DBMS running on a VAX computer under the VMS operating system. This prototype system provides users with the following capabilities:

- SQL queries to IMS
- Distributed SQL queries to IMS and Ingres
- Distributed SQL queries embedded in a C language program

All modules of DATAPLEX are independent of the local database system; the only exceptions are the translator and local DBMS modules, which perform mapping between DATAPLEX and local database systems. Thus, any component system can be incorporated into DATAPLEX through the construction of these two modules. This open and modular architecture permits functionality and performance to be gradually increased.

4.2.3 Mermaid

Mermaid, developed at Data Integration, Inc., is a multidatabase system supporting multiple federated schemas [TBC+87]. It is a front-end system that locates and integrates data maintained by local DBMSs, parts of which may be shared among global users.

A single database view is presented to users of the multidatabase system through two-stage process. First, a federated schema encompassing either complete or partial component databases must be defined. Second, at run time, the federated schema is translated into the actual data storage format.

Users are able to employ a single query language, SQL, to access and integrate data from several databases. The system automatically locates the required data, opens connections to the backend DBMSs, issues queries in the appropriate query languages, and integrates the data from multiple sources. This integration may involve translation into a standard data type, translation of units, combination or division of fields, unification of horizontal fragments, the joining of vertical fragments, and/or the encoding of values.

Mermaid also has the capability to retrieve data from files, which may be defined as a typed object with associated retrieval and display methods. The user selects a set of files of interest by searching on structured fields and displaying structured fields and files in the target list. The report resembles the standard output of a relational system, with files given a symbolic name. New file types can be supported by providing the retrieve and display methods.

Error handling has posed a significant challenge to the developers of Mermaid. Since Mermaid is superimposed on many layers of DBMSs, operating systems, and network protocols, error conditions in any layer may generate errors in other layers. The totality of potential errors and inter-layer responses has not been clearly defined, leaving Mermaid with little guidance in the event of the occurrence of an error.

4.2.4 MULTIBASE

MULTIBASE, developed by Xerox Advanced Information Technology, provides a uniform integrated interface for retrieving data from preexisting, heterogeneous, and distributed databases [SBD+81]. It was designed to allow the user to reference data in such databases with a single query language over one database description (schema). By presenting a globally integrated view of information. MULTIBASE facilitates rapid and straight forward user access to data in multiple databases. The integrated schema and single query language (DAPLEX) simplifies the knowledge necessary on the part of the user.

The MULTIBASE view mechanism is also used to resolve data incompatibilities that frequently arise when separately developed and maintained databases are accessed conjointly. When defining a view, the database administrator applies knowledge of the local databases to predict possible incompatibilities and devise methods for their reconciliation. The methods are then included in the view definition and are followed automatically by the system in generating answers to future queries.

A MULTIBASE prototype system has been implemented in Ada, where it executes on a VAX under the VMS operating system. Local Database Interfaces (LDIs) have been developed for five DBMSs, providing access to Oracle and RIM systems executing under the VMS operating system and to FOCUS, System 2000, and DMR (a hierarchical DBMS developed by the U.S. Army) systems executing under the VMS operating system. To minimize the cost of adding a new DBMS to MULTIBASE, a set of LDI building blocks has been created. These building blocks implement LDI processing that is common to all LDIs and greatly reduce the amount of new software required to create an LDI.

A number of difficulties experienced during the implementation of this project would provide a direction for future enhancements to MULTIBASE. Two important points relate to the challenges of handling local system particularities and the need for automated tools to support the creation and maintenance of MULTIBASE schemas. Automated methods are essential to the administration of the dictionary of a system that integrates databases from many different organizations. Such tools would assist both in creating MULTIBASE schemas and in maintaining consistency as changes occur to the local databases.

4.2.5 TDIE

TDIE is a data integration engine developed at TRW in Redondo Beach, California. Unlike conventional efforts which use a gateway with query language mapping, TDIE has no global data manipulation language. A local DBMS is used on each computer system and a connection to a remote DBMS is triggered when a user issues a demand requiring data integration.

An integrated schema performs uniform query processing for hierarchical, relational, and network data models. TDIE does not support global updates; users at other systems are informed of an update in one local system by polling the potential source files at a rate of their choosing. Types of data that are frequently accessed and must be current are polled at short intervals. This approach eliminates extraneous activity and avoids unnecessary copying of bulk data by transferring only the changes in data.

TDIE is comprised of a data transform manager and a host interface manager which can reside on a single processor or be partitioned across multiple processors. The data transform manager contains the centralized dictionary which defines the global conceptual schema. This knowledge-based dictionary supplies scripts, translation software, and data locations for full data integration. Scripts are instructions for data transfer which are sent to the host interface manager, while the translation software directs the translation from one query language to another. The incorporation of another local DBMS involves the addition of the appropriate translators and scripts to the dictionary. TDIE provides a front-end tool, termed the Integration Advisor, for interactive construction of a global integrating schema.

The host interface manager forms a bridge to the host applications, providing automatic updating and data retrieval. Time-critical applications in transaction processing or realtime control are not supported, although TDIE does support global updates and enables the integration of hierarchical, relational, and network data models.

4.3 A Review of Commercial Systems

In this subsection, we discuss various commercial multidatabase systems. Unlike their counterparts in research institutions, these commercial systems are designed to be used by non-database experts for the maintenance and process of relevant information. They must therefore provide all necessary system capabilities, including user-friendly interfaces, a reliable recovery mechanism, well-written documentation, and complete error handling, and must maintain a high performance standard. These criteria may not be met by many research prototypes.

4.3.1 Ingres/STAR

The Ingres/STAR system allows users to access a distributed database which is defined as a collection of tables from one or more Ingres databases. Any set of tables from any set of Ingres databases can be combined to form a new distributed Ingres/STAR database. Such tables can be obtained not only from databases under an Ingres DBMS but also from databases accessible via an Ingres Gateway and, in the near future, other Ingres/STAR databases. A single Ingres/STAR server may service multiple distributed databases, and multiple Ingres/STAR servers may exist in the network.

Access to an Ingres/STAR distributed database is transparent in that once the distributed database has been created, its users need not be aware of the existence its component Ingres databases. Ingres/STAR appears to front-end programs as if it were a centralized Ingres DBMS. The functioning of front-end programs is unaffected by the distributed nature of the database being accessed.

Ingres/STAR does not itself deal directly with the physical storage and retrieval of data, relying instead on the Ingres/DBMS and/or Ingres/Gateway components for this purpose. The Ingres/STAR component communicates with these Ingres data managers (either Ingres DBMSs or Gateways) in one manner and regarding the same information as a front-end program. Ingres/STAR sends a query language representation of the desired task, and the data manager replies with the requested data.

Some of the available Ingres/Gateways are put into a production mode for RMS files and RDB databases on a VAX computer under the VMS operating system and for DB2 databases on IBM mainframes under the MVS operating system. One of the biggest challenges in designing gateways has been to understand the capabilities required of the local data managers for participation in distributed operations. A subset of SQL that is supported by all gateways has been designed, and this common SQL is used in the Ingres/STAR product. This common subset is expected to evolve over time.

4.3.2 Oracle V5

In its version V5.1.17, Oracle has become a multidatabase management system. It permits the creation of several databases at the same site and the formulation of elementary multidatabase queries. The Oracle multidatabase language is termed SQL*PLUS. The database name (or even the site name) does not prefix the table name but postfixes it after the character @. This capability facilitates the resolution of name conflicts. The user also has a repertoire of statements with which to define aliases for table and database names. This capability, called database linking, should not be confused with the different meaning of this term in the Ingres/STAR system.

The Oracle language also offers statements for interdatabase queries unknown in other commercial systems but largely similar to those of MRDSM. Multidatabase manipulations for distributed databases are also available through a distributed database management component SQL*STAR. It is likely that the latter will become a permanent feature of Oracle, so that it will cease to be a centralized and monodatabase system. Transaction processing in Oracle uses two-phase locking and variants of the two-phase commit protocol. This newest version of the system should be capable of coordinating non-Oracle databases as well as participating in the commit protocols coordinated by other DBMSs.

4.3.3 SUPRA

SUPRA, a product of Cincom, provides a commercial implementation of the ANSI/SPARC three-schema architecture. The initial development of the precursor to the current product was performed in Germany.

The SUPRA system integrates distributed heterogeneous databases and manages distributed processing by placing a locally resident SUPRA kernel within each host machine or database. This has the effect of making the hosts aware of the integrated database network, and, in essence, creates a "standard protocol" through which the databases communicate.

The SUPRA kernel consists of two basic engine components: the distributed relational data manager (DRDM) and the heterogeneous data management processor (HDMP), along with a portion of the global directory. Using the kernel on each local host, the requesting node coordinates the entire unit of work, including the decomposition of the query into separate neutral protocol requests and the determination of where they should be sent. The local/receiving nodes, each with its own kernel, translate these requests into the appropriate local data manipulation languages, gather the data, perform calculations, and return the results to the requesting node.

SUPRA provides for the use of synonyms and the resolution of scale conflicts. The decentralized approach permits the formulation of a synonym table oriented to each user's particular application and working language. The global dictionary has the ability to retain rules for performing scale translation and calculation before data integration. A unique "trigger" feature alerts the user to the possibility of a data conflict.

SUPRA utilizes a two-phase commit procedure which provides good concurrency control. While assuring secure updates for SUPRA and other database management systems which support two-phase commit, the product currently supports updates from join views to SQL-based distributed databases only.

4.3.4 Sybase

Sybase attempts to open the architecture as widely as possible to allow any database, application, or service to be integrated into the client/server architecture in a heterogeneous environment. No global data model or schema is enforced. Rather, distributed operations can be supported via application programming or database-oriented RPCs between SQL Servers, providing a high degree of site autonomy. At the same time, the SQL Server provides full DBMS support at each location and "prepare to commit" support for a two-phase commit protocol that guarantees recoverability for multisite updates.

Sybase is based on the relational model and supports both interactive and programmed access to the SQL Server or any Open Server application. The basic query language is SQL. The SQL Server also supports triggers as independent objects in the database. These triggers have the capabilities of procedures, with these important extensions:

- They cannot be directly executed, but rather are executed as a side effect of an SQL delete, insert, or update.

- A trigger is an extension of the user's current SQL statement. It can roll back or modify the results of a user's transaction.

- Triggers can view the data being changed.

The Sybase Open Server provides a consistent method of retrieving SQL requests or remote procedure calls from an application which is based on the Sybase SQL Toolset or which uses the Sybase Open Client Interface and passing them to a non-Sybase database or application. While Open Client gives users the flexibility to employ a variety of front-end packages or applications for accessing and updating data, the Sybase Open Server allows access to and updating of foreign (non-Sybase) databases and applications.

At run time, an application program issues a database RPC (Remote Procedure Call) to the distributed database system, which may consist of any combination of SQL Servers or Open Servers. If the data are stored in non-Sybase sources, the Open Server provides the necessary data type and network conversions to allow the Open Client to process the returned data.

Sybase supports distributed updates that span multiple locations. A two-phase commit protocol, coded in the application, enforces distributed transaction control across multiple SQL Servers.

5 A Review of Object-oriented Prototypes and Commercial Systems

The benefits of using an object-oriented data model include, but are not limited to:

- a richer data model which closely matches applications,
- typing and inheritance facilities that promote reuse of software components,
- data abstraction and encapsulation,
- facilitating modular development,
- simplified management of policies and protocols, and

- allowing different levels of integration.

An object-oriented approach to the development of multidatabase systems therefore allows these systems to share the advantages of database systems and object-oriented programming languages. For this reason, our survey treats separately those multidatabase systems that use an object-oriented approach to component systems integration. Such systems are also further classified as academic research prototypes, industrial research prototypes, and commercial systems.

5.1 A Review of Academic Research Prototypes

5.1.1 A la carte Framework

The A la carte Framework [DKH92], developed at the University of Colorado, provides an extensible set of reusable components to integrate heterogeneous and persistent object stores. There are three types of functionality required to implement a heterogeneous configuration, and the A la carte Framework provides components to serve each of these functions. Components of the first type are combined to construct facilities, such as heterogeneous transaction management, which are typically implemented as an intrinsic part of a heterogeneous management system (HMS). Components of the second type specify integration protocols which manage the mapping of the implementation of a given function, such as concurrency control, between autonomous systems; these protocols are also typically implemented as part of an HMS itself. These integrating components help ensure correct interoperation between autonomous systems with disparate implementations. Components of the third type are used to extend or "bootstrap" an autonomous system with the functions necessary to operate in a heterogeneous configuration, if such functions are not normally present. In this case, A la carte components are combined with an existing system to create an effectively new autonomous system that has the capabilities required to operate in a heterogeneous architecture. The framework also provides a mechanism for capturing constraints which govern the mixing and interchange of components, reducing the complexity of heterogeneous systems integration. Finally, each component can have different implementations, within certain constraints, so that some tailorable HMS behavior can be achieved.

The A la carte Framework could be used to integrate a spectrum of domains found in heterogeneous systems, including schema representation, query processing, transaction management, and recovery. It defines a multi-dimensional model that helps clarify the process of heterogeneous systems integration from highly applications-oriented standpoint. This framework clarifies for systems integrators the implications of certain design decisions and the alternative architectures that are available to them. By uncovering the internal behaviors of autonomous systems, this framework supports the creation of tailorable heterogeneous configurations. Despite its systems-orientation, its mata-level representation of heterogeneous architectures renders it very flexible and wide applicable.

5.1.2 FBASE

FBASE [Mul92], developed at Purdue University, is a federated object-oriented database system that uses an object-oriented approach to support the integration of database systems, particularly relational, nested relational, extended relational, and object-oriented database systems.

In the federated architecture [HM85], each component system has private, import, and export schemas, and each component system functions as a global query processor. The private schema describes the data of the component system, while the export schema describes the data it may share with others. The import schema describes the data located at remote systems of which the system in question is aware and is derived from the export schemas of the other component systems. FBASE includes FBASE component systems and heterogeneous component systems. In order to preserve autonomy, only the FBASE component systems generally function as traditional federation component systems. Heterogeneous component systems would typically only provide an export schema and could not issue global queries.

Since one of the main goals of FBASE has been to allow flexibility in the method of component system integration, various approaches for component system integration are presented that balance ease of implementation with ease of use and/or functionality. While FBASE does not provide global transaction management, the FBASE system is currently being integrated into the InterBase multidatabase system [BCD+93], which provides support for global transaction management.

5.1.3 Thor

Thor, a distributed object-oriented DMBS developed in the Laboratory for Computer Science at MIT [LDS92], is intended for use in heterogeneous distributed systems to allow programs written in different programming languages to conveniently share objects. Thor objects are persistent in spite of failures, are highly likely to be accessible whenever needed, and can be structured to reflect the kinds of information of interest to users. Thor combines the advantages of the object-oriented approach with those of database systems. Users can store and manipulate objects that capture the semantics of their applications and can also access objects by high-level path names, by navigation, and by means of queries.

Thor is intended to run in a distributed environment in which computing nodes are connected by network. Some of these nodes are Thor servers, which store the objects in the Thor universe, while others are client nodes where users of Thor run their programs. Although it is possible for a single node to act as both server and client, it is most common for client and server nodes to be distinct.

Frontends are run at client nodes, while backends and object repositories are run at server nodes. Users always interact with Thor via a frontend, which typically resides at the user's workstation and makes use of backend or object repositories to carry out client requests. The frontends and backends understand types and perform operations, while the object repositories are concerned only with managing resilient storage for the objects.

All modifications to objects are made at frontends and are installed in phase 2 of a two-phase commit protocol only if phase 1 is successful. An optimistic concurrency control scheme is currently used to maintain the consistency of Thor, although a pessimistic concurrency control is also under consideration. A primary copy scheme is employed to handle replication, and objects are allowed to move throughout the system.

5.2 A Review of Industrial Research Prototypes

5.2.1 Carnot

The Carnot Project [Woe93] at MCC addresses the problem of logically unifying physically-distributed and enterprise-wide heterogeneous information. Specifically, Carnot will provide a user with the means to navigate information efficiently and transparently, to update that information consistently, and to write applications easily for large heterogeneous distributed information systems.

Carot has developed and assembled a large set of generic facilities that are focused on the problem of managing integrated enterprise information. These facilities are organized as five sets of services: communication services, support services, distributed services, semantic services, and access services.

The communication services provide the user with a uniform method of interconnecting heterogeneous equipment and resources. The support services implement basic network-wide utilities that are available to applications and other higher level services. The distribution services support relaxed transaction processors which manage information inconsistently when appropriate. They also provide a distributed agent facility that interacts with client applications, directory services, repository managers, and Carnot's declarative resource constraint base, to build ESS work flow scripts designed to carry out a particular business function. The semantic services provide a global or enterprise-wide view of all the resources integrated within a Carnot-supported system. A suite of tools uses an extensive set of semantic properties to declaratively represent an enterprise information model within the global context and to construct bidirectional mappings between the model and the global context. The access services provide mechanisms for manipulating the other four Carnot services, permitting developers to employ a mix of user interface software and application software to build enterprise-wide systems.

A number of sponsors of the Carnot research project have initiated the development of applications using the Carnot prototype; these include Eastman Kodak, Boeing Computer Services, BellCore, and Ameritech. The focus of these applications varies from an emphasis on heterogeneous database access to a concern with more generalized workflow processing.

5.2.2 CIS

CIS (Comandos Integration System) [BGN+88, BNPS89] provides an object-oriented interface for its component systems. CIS is part of the COMANDOS project (Construction and Management of Distributed Office Systems) under the European Strategic Programme for Research in Information Technology (ESPRIT). CIS integrates various systems, including relational database systems, graphic databases, and public data banks.

CIS supports the creation of integrated applications accessing data managed by heterogeneous and independent application environments residing on various sites throughout a computer network. These applications may run on different hardware and different operating systems and DBMSs, each with its own conventions regarding data representation and storage. CIS models this situation through the concept of (logical) nodes, classified into Client Nodes and Server Nodes, connected by a (logical) communication network.

A Server Node is the abstraction of a representative of CIS within a pre-existing application environment. The role of CIS, in the context of a server, is to make available to clients a uniform object-oriented interface which is superimposed upon the local environment. The CIS component that resides on the server is called Cis_Server.

A Client is the abstraction of an application, based on CIS, which accesses the functions provided by one or more services. A client performs an integrated application function using data from different pre-existing environments, thereby implementing the integration goal of CIS. The CIS component that resides on the client is called Cis_Client.

An object-oriented approach is used for schema translation. CIS Servers have been designed to provide an object-oriented programming environment for the implementation of abstract classes.

Questions of schema integration and transaction management are not addressed, and distribution transparency is not provided by the system.

5.2.3 DOM

The DOM (Distributed Object Management) project [BÖH+92] at GTE Laboratories supports application development in a distributed object-oriented environment that integrates autonomous and heterogeneous database and non-database systems (such as file systems). The DOM system supports schema translation to a common object-oriented data model.

A DOM system consists both of *native* objects that are fully implemented by the DOM components of the system and of objects that are wholly or partially implemented in heterogeneous attached systems. These attached systems are not limited to database systems but may include conventional file systems, hypermedia systems, or application programs. Interaction with an attached system occurs through objects defined in a DOM's Local Application Interface (LAI). Objects from the attached systems have placeholders defined within the DOM object space. These placeholders are used for the materialization of external data within DOM, data transfer, the invocation of application processing of external data, and global concurrency control. When used for concurrency control purposes, LAI objects can be treated like any other DOM object. The LAI objects act as guards to control across a DOM/attached system boundary. If they are defined as *active* objects, they can enforce cross-boundary consistency concepts.

The DOM system provides an advanced transaction model that supports both *closed nested* and *open nested* transactions and combinations thereof. While closed nested transactions are equivalent to traditional nested transactions, open nested transactions do not provide the same top-level atomicity, so the partial results of the transaction may be viewed by other transactions. Compensating, contingency, and non-vital subtransactions are allowed, and execution dependencies may be specified between subtransactions.

A correctness theory and implementation approach has not been developed for the DOM transaction model. Since the component systems to be integrated may be non-database systems that may not support transactions, it would be impossible to guarantee correct global transaction management. The degree of correctness provided is therefore dependent on the capabilities of the systems integrated.

5.2.4 EIS/XAIT

EIS/XAIT (Engineering Information System/Xerox Advanced Information Technology) [PSH91] is a framework that uses an object-oriented approach to integrating heterogeneous systems.

An object-oriented interoperability framework, termed the Object Management System (OMS), is employed to ensure interoperability by:

- Providing build-in abstract data types and constructors for user-defined abstract data types that permit the heterogeneity of component systems to be masked.
- Providing reference resolution, type checking, and a function resolution mechanism to reference and operate over objects from heterogeneous systems; and
- Providing various execution-related primitives, including transaction management support and exception handling.

A query language is provided that consists of functions that operate on collections of types, including the following functions:

- union, intersection, and difference;
- select, project, and join;
- flattening and grouping functions for sets with embedded structures.

While nested transactions are possible, their level of correctness is depend on the capabilities supported and externalized by the component systems. For example, two-phase commitment is provided only when component systems support this capability by providing a visible prepare-to-commit state.

5.2.5 OIS

OIS (Operational Integration System) [GCO90] provides applications with a uniform object-oriented interface for accessing data managed by heterogeneous information systems. Conceptually, OIS can be depicted as a three-layer system. At the abstract level, the programmer is presented with an abstract view of objects, which are described in terms of classes and user-defined operations. This uniform method of reality description, termed the "OIS Integration Data Model (IDM)", permits distributed applications to be coded in an object-oriented manner. A set of predefined operations which locate and access objects is associated with these IDM classes. Such generic operations constitute the Operation Interface Abstract Level. At the Operation Interface Implementation Level, the code implementing generic operations is superimposed on the application environments to be integrated. Those programs may reside in local systems and therefore be pre-existing and possibly heterogeneous and make use of local data management system functionalities. This process, which is termed operational mapping, may cause a generic operation to have several implementations. Although operational mapping is less automated than schema mapping, it does not require mapping between the data elements manipulated by the local data management system and some "global schema". The introduction of an abstract level enhances the software reusability.

OIS models the heterogeneous world through the concept of (logical) nodes, classified into Client nodes and Server nodes, connected by a (logical) communication network. A Server is an autonomous and independent environment which groups together a set of objects under common application semantics. The role of OIS in the context of a Server (OIS_Server) is to make applications available to the clients. Therefore, a Server is the partial abstraction of a pre-existing application environment.

A Client is the abstraction of a distributed application based on OIS. It is composed of the distributed application and of the OIS_Client (the client portion of OIS) which encapsulates the IDM, together with the abstract view of OIS_Server services. Therefore, the Client implements the integration goal of OIS.

OIS provides both a query language interface, called QL, as well as a method for embedding OIS functionality within C++ programs. OIS translates component schemas into its own data model and does not require component system modification.

5.2.6 Pegasus

Pegasus [ASD$^+$91] is a heterogeneous multidatabase system under development at Hewlett Packard Laboratory. The goal of Pegasus is to integrate distributed object-oriented, relational, and other information systems. Pegasus uses an object-oriented model and language for its data model and query language and addresses the issues of both schema translation and schema integration.

Pegasus exploits object-oriented data modeling and programming capabilities, employing both type and function abstractions to deal with mapping and integration problems. Function implementation can be defined in an underlying database language or a programming language. Data abstraction and encapsulation facilities in the Pegasus object model provide an extensible framework for approaching heterogeneities in both traditional database systems and nontraditional data sources ranging from simple text to complex multimedia systems.

The Pegasus data model is based on that of Iris [WLH90] and includes three basic constructs: objects, types and functions. Pegasus provides HOSQL (Heterogeneous Object SQL) as a common query language. HOSQL is a functional as well as an object-oriented language that employs declarative statements to manipulate multiple heterogeneous databases and to create types, functions, and objects in both Pegasus and underlying local databases. If desired, specifications of types and functions can also be imported from underlying local databases and can then be integrated into the Pegasus native schemas. The HOSQL language offers transparent and explicit remote access. As Pegasus integrates information systems that do not support transactions, it does not consider global transaction management.

5.2.7 ViewSystem

ViewSystem [KDN90] is part of the KODIM (Knowledge Oriented Distributed Information Management) project at the GMD-IPSI national research institute. ViewSystem uses the VODAK data model to integrate heterogeneous and autonomous information and publication systems. Both schema translation (to the VODAK model) and schema integration are supported.

The VODAK data model provides support for specification of the following concepts:

- Category Generalization constructs a generalized model from classes processing disjoint sets of entities;
- Role Generalization constructs a generalized model from classes that have overlapping sets of entities;
- Specialization;
- Aggregation;
- Grouping.

The constructs underlying these concepts form the basis for system integration in ViewSystem. An object-oriented query language is provided for multidatabase access, and a hybrid query evaluation scheme is proposed which uses both integrated view materialization and query decomposition/translation to process global queries. Transaction management is not discussed within the ViewSystem approach.

5.3 A Review of Commercial Systems

5.3.1 UniSQL/M

UniSQL/M is a multidatabase system from UniSQL, Inc. that integrates multiple UniSQL/M unified relational and object-oriented databases and multiple relational databases. UniSQL/M currently offers drivers (gateways) for INGRES and ORACLE, and plans to support additional relational database drivers in the near future. UniSQL/M is a complete database system and UniSQL/M users can query and update the global database in the powerful SQL/X database language, which is a superset object-oriented extension of ANSI SQL.

UniSQL/M maintains the global database as a collection of views defined over relations in local RDBs and classes in local UniSQL/X databases. UniSQL/M also maintains a directory of the local database relations and classes, that have been integrated into the global database, along with their attributes, data types, and methods. Drawing upon the information in this directory, UniSQL/M translates queries and updates into equivalent forms to be processed by local database systems that manage the relevant data. The local database drivers pass these translated queries and updates to local database systems and transfer the results back to UniSQL/M for format translation, merging, and any necessary postprocessing (e.g., sorting, grouping, and joining). UniSQL/M supports "distributed transaction management" over local databases, through which all updates issued within one UniSQL/M transaction, even if propagated to multiple local databases, are simultaneously committed or aborted.

6 Summary

Tables 1 through 3 summarizes the main features of the systems discussed in this paper, following the taxonomic schema proposed in Section 3. In each table, object-oriented systems follow those using conventional integration methods. In general, a "No" entry indicates either that the capability in question was specifically excluded or was ambiguous, while an "Any" indicates that various local data models are either supported or that their support can be easily implemented.

System Integrated. This column lists the types of component systems that can be integrated into the specified multidatabase system. Type 1 multidatabase systems, which integrate only database systems, are the most common; a few Type 2 schema also integrate non-database systems, a highly complex task requiring the ability to perform transformations among component database and non-database systems. Such systems, while challenging to develop, are more powerful than Type 1 multidatabase systems.

Provision of Schema Translation and Integration. This column indicates whether or not a specified multidatabase system provides a common data model at the global level and/or the facilities to resolve semantic conflicts among component system data. The common data model facilitates data transfer among component systems. For example, data to be transferred from

System	Systems Integrated	Providing Schema Translation & Integration	Local Data Model Supported	Supporting Transactions	Maintaining Component System Autonomy
IMDAS	Various DB and Non-DB Systems	ST[1]	Any	No	Yes
InterBase	Various DB and Non-DB Systems	ST[1]	Any	Yes[2]	Yes
NDMS	Database Systems	ST[1]	Relational & Network	No	Yes
MRDSM	Database Systems	ST[1]	Relational	No	Yes
OMNIBASE	Database Systems	ST[1]	Any	Yes[2]	Yes
PRECI*	Database Systems	ST[1]	Any	Yes[4]	Yes
A la carte	Various DB and Non-DB Systems	No	Any	Yes[3]	No
FBASE	Database Systems	Yes	Relational	No	Yes
Thor	Object Repositories of Thor	No	Object-oriented	No	Yes

[1]Provides schema translation only.

[2]Open and closed nested transactions are provided, but correctness is dependent upon the capabilities of component systems.

[3]Correctness is dependent upon the capabilities of component systems.

[4]Base relation only; replicates are updated by broadcast.

Table 1: Academic Multidatabase Prototype Systems

System	Systems Integrated	Providing Schema Translation & Integration	Local Data Model Supported	Supporting Transactions	Maintaining Component System Autonomy
ADDS	Database Systems	ST[1]	Rel., Network & Hierarchical	Yes[2]	Yes
DATAPLEX	Database Systems	ST[1]	Relational & Hierarchical	No	Yes
Mermaid	Various DB and Non-DB Systems	ST[1]	Any	NO	Yes
MULTIBASE	Database Systems	Yes	Any	NO	Yes
TDIE	Database Systems	Yes	Rel., Network & Hierarchical	Yes[4]	Yes
Carnot	Various DB and Non-DB Systems	ST[1]	Any	Yes[5]	Yes
CIS	Various DBSs	ST[1]	Any	No	Yes
DOM	Various DB and Non-DB Systems	ST[1]	Any	Yes[5]	Yes
EIS/XAIT	Engineering Info. Systems	ST[1]	Any	Yes[6]	Yes
OIS	Heterogeneous Info. Systems	ST[1]	Any	No	Yes
Pegasus	Heterogeneous Info. Systems	Yes	Any	No	Yes
ViewSystem	Information and Publication Systems	Yes	Any	No	Yes

[1]Provides schema translation only.
[2]If local data items can be locked by global transactions.
[3]If a two-phase commit protocol is supported.
[4]Based on "ownership".
[5]Open and closed nested transactions are provided, but correctness is dependent on the capabilities of component systems.
[6]Nested transactions are provided, but correctness is dependent on the capabilities of component systems.

Table 2: Industrial Multidatabase Prototype Systems

System	Systems Integrated	Providing Schema Translation & Integration	Local Data Model Supported	Supporting Transactions	Maintaining Component System Autonomy
Ingres/STAR	Database Systems	Yes	Any	Yes[1]	Yes
Oracle V5	Database Systems	Yes	Any	Yes[1]	Yes
SUPRA	Database Systems	Yes	Relational & Network	Yes[1]	Yes
Sybase	Database Systems	Yes	Any	Yes[1]	Yes
UniSQL/M	Ingres, Oracle, and UniSQL/X	Yes	Relational & Object-oriented	Yes[2]	Yes

[1]Two phase-commit protocol must be supported.
[2] Correctness is dependent on the capabilities of component systems.

Table 3: Commercial Multidatabase Systems

system 1 to system 2 will first be transformed from the system 1 format to that of the common data model, when it enters the data flow; upon arrival at system 2, the data is further transformed into the format of that system. Such a common data model resolves inconsistencies among the presentation formats of similar data. However, a new component system cannot be integrated into a multidatabase system until a mapping has been provided between the common data model and the data model used by the component system.

Schema integration addresses such semantic conflicts as naming, type, and unit conflicts. For example, two component systems may represent employee salaries in dollars or pounds, respectively, a unit conflict which must be resolved when data is transfered from one component system to another. The provision of schema integration cannot, however, enable a multidatabase system to automatically resolve all varieties of semantic conflicts among any component systems, as such conflicts can assume a virtually infinite variety of permutations.

Local Data Model Supported. This column indicates those types of local data models which are supported by a specified multidatabase system. Possible data models employed by component systems include relational, network, hierarchical, entity-relationship, and object-oriented models. A non-database component system may use any format to store and process data. The flexibility of a multidatabase system is enhanced by its ability to support a variety of local data models.

Supporting Transactions. This column indicates whether a specified multidatabase system semantically supports ACID (atomicity, consistency, isolation, and duration) transactions. While a close nested transaction must maintain the ACID properties syntactically, an open nested transaction is not required to provide top-level atomicity, and the partial results of such a transaction may be viewed by other transactions. In this case, compensating transactions can be specified to undo the effects of a committed transaction, and contingency transactions can be executed if a given transaction fails.

Maintaining Component System Autonomy. If a multidatabase system requires any degrees of modification of its component systems, the local autonomy of its component systems is not preserved. Even a minor modification to a software system may not be possible if the owner of a component system may not have a source code version of the database or the resources to modify it even if this code version was available; also, the component system may belong to a different organization than the one developing the multidatabase system, and that organization usually does not grant permissions to others for the modifications of their products.

The overview of multidatabase systems presented in this paper was constrained by the lack of detail regarding implementation and performance provided in published descriptions of such systems. Few references discussed transaction management and query processing aspects which are strongly affected by the integration scheme used. The information-hiding effects of behavioral mapping, while aiding system integration, may present difficulties for transaction management and query processing. Since the bulk of the research to date on schema integration has involved only structural mapping, [Ber91] is correct in suggesting that more work needs to be done on schema integration involving behavioral mapping.

Tables 1 through 3 indicate more than half of surveyed systems integrate both database systems and various non-database systems. Most surveyed systems provide a common data model to map among heterogeneous component systems which translates moved or shared data into the appropriate format. As most surveyed systems integrate various non-database systems, a multidatabase system must be able to perform such mapping between data models of any type, a condition which is fulfilled in most surveyed systems. Only about a third of surveyed systems provide schema integration facilities, indicating that schema integration is either difficult to implement or of practical unimportance. About two thirds of surveyed systems provide limited transaction support, indicating that no satisfactory solution has been found for this issue. All but one surveyed systems maintain the autonomy of component systems; such autonomy has clearly become a common goal of multidatabase system developers. Nearly half of surveyed systems are object-oriented, indicating that multidatabase developers are focusing increasingly on an object-oriented approach.

The integration of heterogeneous information systems has become a critical goal for many organizations, one which is being pursued by many different groups with varying approaches. In this paper, we have proposed a new taxonomy that characterizes these approaches by five criteria: systems integrated, provision of schema translation and integration, support of local schemas, support of transaction management, and maintenance of component system autonomy. This taxonomy has been employed here to characterize twenty-six actual multidatabase systems which represent the broad spectrum of approaches being pursued worldwide, including both commercial products and prototype systems. These multidatabase systems and the approaches they represent are harbingers of the capabilities that will become widely available in coming years.

References

[AGMS87] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency control and recovery for global procedures in federated database systems. In *IEEE Data Engineering Bulletin*, pages 5–11, September 1987.

[ASD+91] R. Ahmed, P. De Smedt, W. Du, W. Kent, M.A. Ketabchi, W.A. Litwin, A. Rafii, and M.C. Shan. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, 24(12):19–27, December 1991.

[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).

[Ber91] Elisa Bertino. Integration of heterogeneous data repositories by using object-oriented views. In *Proceedings of the IEEE First International Workshop on Interoperability in Multidatabase Systems*, pages 22–29, Kyoto, Japan, April 1991.

[BGMS92] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *The VLDB Journal*, 1(2):181–239, October 1992.

[BGN+88] E. Bertino, R. Gagliardi, M. Negri, G. Pelagatti, and L. Sbattella. The comandos integration system : an object oriented approach to the interconnection of heterogeneous applications. In *Proceedings of the Second International Workshop on Object-Oriented Database Systems*, pages 213–218, Bad Münster am Stein-Ebernburg, FRG, September 1988.

[BGRS91] Y. Breitbart, D. Georgakopolous, M. Rusinkiewicz, and A. Silberschatz. On Rigorous Transaction Scheduling. *IEEE Transactions on Software Engineering*, 17(9):954–960, 1991.

[BHP92] M. W. Bright, A. R. Hurson, and Simin H. Pakzad. A taxonomy and current issues in multidatabase systems. *Computer*, 25(3):50–60, March 1992.

[BL84] C. Batini and M. Lenzirini. A methodologies for data schema integration in entity-relationship model. *IEEE Transactions on Software Engineering*, 10(6), November 1984.

[BLN86] C. Batini, M. Lenzerini, and S.B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):232–364, December 1986.

[BNPS89] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Integration of heterogeneous database applications through an object-oriented interface. *Information Systems*, 14(5):407–420, 1989.

[BÖH+92] A. Buchmann, M. T. Özsu, M. Hornick, D. Georgakopoulos, and F. A. Manola. A transaction model for active distributed object systems. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.

[BS88] Y. Breitbart and A. Silberschatz. Multidatabase update issues. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 135–142, June 1988.

[BST90] Y. Breitbart, A. Silberschatz, and G. Thompson. Reliable transaction management in a multidatabase system. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 215–224, May 1990.

[BT85] Y.J. Breitbart and L.R. Tieman. Adds - heterogeneous distributed databse system. *Distributed Data Sharing Systems*, pages 7–24, 1985.

[CAC91] Special Section on Next-generation Database Systems. *Communications of the ACM*, 34(10), October 1991.

[CBE93] Jiansan Chen, Omran A. Bukhres, and Ahmed K. Elmagarmid. IPL: A Multidatabase Transaction Specification Language. In *Proc. of the 13th International Conference on Distributed Computing Systems*, Pittsburgh, PA, May 1993.

[Chu90] C.W. Chung. DATAPLEX: An Access to Heterogeneous Distributed Databses. *Communications of ACM*, 33(1):70–80, January 1990.

[DAODT85] S.M. Deen, R.R. Amin, G.O. Ofori-Dwumfuo, and M.C. Taylor. The Architecture of a Generalized Distributed Database System - PRECI*. *Computer Journal*, 28(3):282–290, 1985.

[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.

[DEK91] W. Du, A. Elmagarmid, and W. Kim. Maintaining transaction consistency in multidatabases using quasi serializable executions. In *Proceedings of COMPCON SPRING'91*, pages 152–155, San Francisco, California, 1991.

[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.

[DH84] U. Dayal and H.Y. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Tran. on Software Engineering*, 10(6):628–644, 1984.

[DKH92] P. Drew, R. King, and D. Heimbigner. A toolkit for the incremental implementation of heterogeneous database management systems. *The VLDB Journal*, 1(2):241–284, October 1992.

[ea85] W. Staniszkis et al. Architecture of the Network Data Management System. In *Proceedings of the 3rd Int'l Seminar on Distributed Data Sharing Systems*, Amsterdam, North-Holland, 1985.

[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.

[GCO90] R. Gagliardi, M. Caneve, and G. Oldano. An operational approach to the integration of distributed heterogeneous environments. In *Proceedings of the PARBASE-90 Conference*, pages 368–377, Miami Beach, Florida, March 1990.

[GM83] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

[GPZ86] V. Gligor and R. Popescu-Zeletin. Transaction management in distributed heterogeneous database management systems. *Information Systems*, 11(4):287–297, 1986.

[Gra78] J. N. Gray. Notes on Database Operating Systems. In *Operating Systems: An Advanced Course, Lecture Notes in Computer Science*, pages 624–633. Springer-Verlag, Berlin, 1978.

[GRS91] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On serializability of multidatabase transactions through forced local conflicts. In *Proceedings of the 7th Intl. Conf. on Data Engineering*, pages 314–323, Kobe, Japan, April 1991.

[HF83] M.J. Han and P.S. Fisher. The problems of data structure and application software coonversion in a heterogeneous environment. In P.S. Fisher, J. Slonim, and E.A. Unger, editors, *Advances in Distributed Processing Management*, volume 2, pages 145–178. Wiley, Chichester, West Sussex, England, 1983.

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

[KDN90] M. Kaul, K. Drosten, and E. J. Neuhold. ViewSystem: Integrating heterogeneous information bases by object-oriented views. In *Proceedings of the Sixth International Data Engineering Conference*, pages 2–10, Los Angeles, California, USA, February 1990.

[KLS90] H. Korth, E. Levy, and A. Silberschatz. A Formal Approach to Recovery by Compensating Transactions. In *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia, August 1990.

[KM89] M. Kuntz and R. Melchert. Pasta-3's Graphical Query Language: Direct Manipulation, Cooperative Queries, Full Expressive Power. In *Proceedings of the International Conference on Very Large Data Bases*, pages 97–106, Amsterdam, The Netherlands, August 1989.

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

[KSL+87] V. Krishnamurthy, S. Y. M. Su, H. Law, M. Mitchell, and E. Barkmeyer. A distributed Database Architecture for an Integrated manufacturing Facility. In *Proceedings of the International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, Oct 1987.

[Lar83] J. A. Larson. Bridging the gap between network and relational database management systems. *Computer*, pages 82–92, September 1983.

[LDS92] B. Liskov, M. Day, and L. Shrira. Distributed object management in thor. In *Proceedings of the 3rd International Workshop on Object-Oriented Database Systems*, pages 1–15, Edmonton, Alberta, Canada, July 1992.

[Lit85] W. Litwin. An Overview of the Multidatabase System MRDSM. In *Proceedings of the ACM Annual Conference*, pages 524–533, Denver, Colorado, 1985.

[LNE89] J. A. Larson, S. B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transaction on Software Engineering*, 15(4):449–463, April 1989.

[MB81] A. Motro and P. Buneman. Constructing superviews. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 56–64, Ann Arbor, Mich., USA, April 1981.

[MRB+92] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. The concurrency control problem in multidatabases: Characteristics and solutions. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 288–297, 1992.

[Mul92] James G. Mullen. FBASE: A federated objectbase system. *International Journal of Computer Systems Science and Engineering*, 7(2):91–99, April 1992.

[NEL86] S. Navathe, T. Elmasri, and J. Larson. Integrating user views in database design. *IEEE Computers*, 19(1):50–62, January 1986.

[NSE84] S. Navathe, S. Sashidhar, and T. Elmasri. Relationship merging in schema integration. In *Proceedings of 10th VLDB*, August 1984.

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

[PSH91] Girish Pathak, Bill Stackhouse, and Sandra Heiler. EIS/XAIT project: An objectbase interoperability framework for heterogeneous systems. *Computer Standards and Interfaces*, 13:315–319, 1991.

[Pu88] C. Pu. Superdatabases for composition of heterogeneous databases. In *Proceedings of the International Conference on Data Engineering*, pages 548–555, February 1988.

[REC$^+$89] M. Rusinkiewicz, R. Elmasri, B. Czejdo, D. Georakopoulous, G. Karabatis, A. Jamoussi, L. Loa, and Y. Li. Omnibase: Design and implementation of a multidatabase system. In *First International Symposium in Parallel and Distributed Processing*, pages 162–169, Dallas, Texas USA, 1989.

[Row85] L.A. Rowe. Fill-in-the-Form Programming. In *Proceedings of 11th International Conference on Very Large Data Bases*, Stockholm, Sweden, August 1985.

[SBD$^+$81] J.M. Smith, P.A. Bernstein, U. Dayal, N. Goodman, T. Landers, K. Lin, and E. Wong. Multibase: Integrating heterogeneous distributed databases systems. In *Proceedings of AFIPS Conference*, pages 487–499, 1981.

[SG89] A. Sheth and S. Gala. Attribute relationships: An impediment in automating schema integration. In *Workshop on Heterogeneous Database Systems*, Chicago, IL, December 1989.

[SL88] A.P. Sheth and J.A. Larson. A tool for integrating conceptual schemas and user views. In *Proceedings of 4th International conference on Data Engineering*, 1988.

[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.

[TBC$^+$87] M. Templeton, D. Brill, A. Chen, S. Dao, E. Lund, R. McGregor, and P. Ward. Mermaid – a front-end to distributed hetergeneous databases. *Special Issue on Distributed Database System, Proceedings of the IEEE*, 75(5), May 1987.

[TTC$^+$90] Gomer Thomas, Glenn R. Thompson, Chin-Wan Chung, Edward Barkmeyer, Fred Carter, Marjorie Templeton, Stephen Fox, and Berl Hartman. Heterogeneous Distributed Database Systems for Production Use. *ACM Computing Surveys*, 22(3), September 1990.

[WLH90] Kevin Wilkinson, Peter Lyngbæk, and Waqar Hasan. The iris architecture and implementation. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):63–75, March 1990.

[Woe93] D. Woelk. Using Carnot for Enterprise Information Integration (Synopsis). In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, pages 133–136, San Diego, California, January 1993.

[ZE93] Aidong Zhang and Ahmed K. Elmagarmid. On global transaction scheduling criteria in multidatabase systems. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, pages 117–124, San Diego, California, January 1993.