A Taxonomy and Current Issues in Multidatabase Systems

M.W. Bright, A.R. Hurson, and Simin H. Pakzad
Pennsylvania State University

Global access and local autonomy are central to multidatabase system design. This article draws on other information-sharing systems research to define key issues.

Information is a key resource in the daily operation of business, government, and academic organizations. Today, organizational information is frequently represented in computer databases. As organizations and users become more sophisticated, the sharing of information resources also increases. For example, the French Teletel system currently gives 1.8 million users access to more than 1,500 separate databases.

However, a multitude of systems usually means multiple access methods and user paradigms. It is unreasonable to ask users to learn all these access methods, yet it is also unreasonable to expect organizations to convert all their systems to a single common data model with a single access method. Multidatabase systems give users a common interface to multiple databases, while minimizing the impact on existing database operations.

Database systems often serve critical functions and represent significant capital investment. Many organizations have several different computers and database systems. In many cases, this environment must be preserved while also addressing the need to share information on a more global basis. Integrated access is required to semantically similar information at different nodes and with different data representations. Multidatabases typically integrate information from preexisting, heterogeneous local databases in a distributed environment and present global users with transparent methods to use the total information in the system. A key feature is the autonomy that individual databases retain to serve their existing customer set.

Multidatabases are an important area of current research, as evidenced by the number of projects in both academia and industry. The trade press has also documented the need for user-friendly global information sharing. The next level of computerization will be distributed global systems that can share information from all participating sites. Multidatabases are a key component of this advancing technology.

This article presents a taxonomy of global information-sharing systems and discusses where multidatabase systems fit in the spectrum of solutions. We use this taxonomy as a basis for defining multidatabase systems, then we discuss the issues associated with them. We pay particular attention to the two major design

0018-9162/92/0300-0050$03.00 © 1992 IEEE

approaches for multidatabases — global schema systems and multidatabase language systems. We list and briefly characterize existing multidatabase projects and conclude with a discussion of areas for further research.

What is a multidatabase?

Taxonomy of global information-sharing solutions. There is a wide range of solutions for global information sharing in a distributed system, and the literature uses a variety of terms to describe them. The terms include distributed databases, multidatabases, federated databases, and interoperable systems. All these terms describe a distributed system that includes a global component to access globally shared information and multiple local components that manage only information at that site. The distinctions are in the structure of the global component and how it interacts with local components. Current usage of these terms varies somewhat between authors. Therefore, we first present a taxonomy to distinguish more concretely between systems.

Our taxonomy orders the solutions according to how tightly the global system integrates the local database management systems. A tightly coupled system means the global functions have access to low-level internal functions of the local DBMS. This allows close synchronization among sites and efficient global processing (possibly at the expense of local efficiency). However, it also implies that global functions may have priority over local functions and, therefore, that local DBMSs do not have full control over local resources. In a more loosely coupled system, the global functions access local functions through the DBMS's external user interface. Global synchronization and efficiency are not as high as in the tightly coupled case, but local DBMSs have full control over local data and processing (site autonomy). In the most loosely coupled system, there are few global functions (just raw data exchange) and the local interface to global information is through applications residing above the local DBMS user interface. The definitions below begin with the most tightly coupled systems and proceed to the most loosely coupled. Table 1 shows the major distinctions between classes.

Distributed databases. A distributed database is the most tightly coupled global information-sharing system. Global and local functions share low-level internal interfaces and are so tightly integrated that there is little distinction between them. Distributed databases, therefore, are typically designed in a top-down fashion, with global and local functions implemented simultaneously. The local DBMSs are typically homogeneous (with respect to the data model implemented) and present the same functional interfaces at all levels, even though they may be implemented on different machines. The global system has control over local data and processing. The system typically maintains a global schema created by integrating the schemas of all the local DBMSs. A schema is a structured description of the information available in a database. Global users access the system by submitting queries over the global schema. Because they are so tightly integrated, distributed databases can closely synchronize global processing. Furthermore, since the global functions have complete control over local functions, processing can be optimized for global requirements. As a result, distributed databases have good global performance, but at the cost of significant local modification and loss of control. Ceri and Pelagatti provide a good introduction to distributed databases.[1]

Global schema multidatabases. Global schema multidatabases are more loosely coupled than distributed databases because global functions access local information through the external user interface of the local DBMS. However, the global system still maintains a global schema, so the local sites must cooperate closely to maintain the global schema. Global schema multidatabases are typically designed bottom-up and can integrate preexisting local DBMSs without modifying them. They

Table 1. Taxonomy of global information-sharing systems.

Class	Level of Global Interface to Local DBMS	Local Node Types	Full Global Database Function	Method of Global Integration
Distributed database	Internal DBMS functions	Homogeneous databases	Yes	Global schema
Global schema multidatabase	DBMS user interface	Heterogeneous databases	Yes	Global schema
Federated database	DBMS user interface	Heterogeneous databases	Yes	Partial global schemas
Multidatabase language system	DBMS user interface	Heterogeneous databases	Yes	Access language functions
Homogeneous multidatabase language system	DBMS user interface plus some internal DBMS functions	Homogeneous databases	Yes	Access language functions
Interoperable systems	Application on top of the DBMS	Any data source that meets the communication protocol	No	No global integration

also normally integrate heterogeneous local DBMSs. This heterogeneity may mean different data models or different implementations of the same data model. Thus, creating the global schema is a more difficult problem than in a distributed database, where the local DBMSs are homogeneous and the global database administrator (DBA) has control over the local schema input to the global schema.

Federated databases. Federated databases are a more loosely coupled subset of global schema multidatabases. There is no single global schema: Each local system maintains a local import and export schema. The export schema is a description of the information the local node is sharing with the global systems. The import schema is a description of the information (both data representation and data origin) from remote nodes that may be accessed locally. Each import schema is in essence a partial global schema. Therefore, each node must cooperate closely only with the specific nodes it accesses. User queries are restricted to local data and the data represented in the local import schema.

Multidatabase language systems. Multidatabase language systems are more loosely coupled than the previous classes because no (partial) global schema is maintained. The global system supports all global database functions by providing query language tools to integrate information from separate databases. User queries can specify data from the local schema of any node participating in the system. Language tools include a global name space and special functions to map information from different models and representations to a model and representation meaningful to the user. Like global schema multidatabases, multidatabase language systems integrate preexisting, heterogeneous local DBMSs without modifying them.

Homogeneous multidatabase language systems. Homogeneous multidatabase language systems are a degenerate form of multidatabase language systems. This subset merits its own class because there are a number of existing multidatabase projects that currently support only homogeneous local DBMSs, and these include some of the first commercial multidatabase systems. The commercial products tend to have limited language functions compared with the projects in the previous class. Some of these systems are actually rather tightly coupled because they allow some global/local interaction below the standard user interface. However, these exceptions are infrequent. Because of the exceptions, members of this class exhibit some characteristics of distributed databases as well as multidatabases.

Interoperable systems. Interoperable systems are the most loosely coupled information-sharing systems. Global function is limited to simple data exchange and does not support full database functionality. Standard protocols are defined for communication among the nodes. Because the global system is not database oriented, local systems may include other types of information repositories, such as expert systems or knowledge-based systems. Interoperable systems are still mainly in the research stage.

Definition of multidatabases. For the purposes of this article, the generic term multidatabase will include the classes of global schema multidatabase, federated database, multidatabase language system, and homogeneous multidatabase language system. A multidatabase is a distributed system that acts as a front end to multiple local DBMSs or is structured as a global system layer on top of local DBMSs. The global system provides full database functionality and interacts with local DBMSs at their external user interface. Although each local node must maintain some global function to interface with the global system, the local DBMSs are autonomous. The global system provides some means (global schema or multidatabase language) of resolving the differences in data representation and function between local DBMSs. This resolution capability is necessary because the same information may be maintained at multiple locations in different forms. The global user can access information from multiple sources with a single relatively simple request.

Issues in multidatabases

Site autonomy. A key aspect of multidatabases, as opposed to distributed databases, is that each local DBMS retains complete control over local data and processing. This is called site autonomy.[2] Each site independently determines what information it will share with the global system, what global requests it will service, when it will join the multidatabase, and when it will stop participating in it. The DBMS itself is not modified by joining the multidatabase. Neither global changes, such as addition and deletion of other sites, nor global optimization of data structures and processing methods has any effect on the local DBMS. Local DBAs are free to optimize local data structures, access paths, and query-processing methods to satisfy local user requirements rather than global requirements. Since the global system interfaces with the local DBMS at the user level, the local DBMS sees the global system as just another local user. Note that site autonomy applies to the local DBMS rather than the local system as a whole. The local system must support some subset of the global function.

The multidatabase approach of preserving site autonomy may be desirable for several reasons. Some local databases may have critical roles in an organization, and it may be impossible from an economic standpoint to change these systems. Site autonomy means the local DBMS can add global access without changing this existing local function. Another economic factor is that an organization may have significant capital invested in existing hardware, software, and user training. All of this investment is preserved when joining a multidatabase because existing local applications can continue operating unchanged. Site autonomy can also act as a security measure because the local DBMS has full control over who accesses local resources through the multidatabase interface and what processing options will be allowed. In particular, a site can protect information by not including it in the local schema that is shared with the global system. An organization's requirement for global access may be minimal or sporadic. Site autonomy allows the local DBMS to join and quit the multidatabase with minimal local impact.

Despite the desirable aspects of site autonomy, it places a large burden on global DBAs. Each site has independent local requirements and makes independent local optimizations to satis-

fy those requirements. Because of this independence and the possibly large number of participating sites, global requirements and desirable global optimizations (of global data structures, access paths, query-processing methods, etc.) are likely to conflict with local ones. The global DBA must work around these conflicts in initial global system design and ongoing global maintenance. Because of the heterogeneity of local DBMSs, the global system may have to dedicate global resources to compensate for any missing local function or information. Some of these problems may be alleviated somewhat if the local DBAs agree to cooperate and conform to some global standards.

Differences in data representation. There are many ways to model a given real-world object (or relationships to other objects), depending on how the model will be used.[3] Because local databases are developed independently with differing local requirements, a multi database system is likely to have many different models, or representations, for similar objects. However, a global user desires an integrated presentation of global information without duplications or heterogeneity.

Name differences. Local databases may have different conventions for naming objects, leading to the problems of synonyms and homonyms. Synonym means the same data item has different names in different databases. The global system must recognize the semantic equivalence of the items and map the differing local names to a single global name. Homonym means different data items have the same name in different databases. The global system must recognize the semantic difference between items and map the common names to different global names.

Format differences. Format differences include differences in data type, domain, scale, precision, and item combinations. For example, a part number may be defined as an integer in one database and as an alphanumeric string in another. Sometimes data items are broken into components in one database while the combination is recorded as a single quantity in another.

Multidatabases typically resolve format differences by defining transformation functions between the local and global representations. Some functions may be simple numeric calculations such as converting square feet to acres. Others may require tables of conversion values or algorithmic transformations. A problem in this area is that the local-to-global transformation may be simple, but the inverse transformation (required if updates are supported) may be very complex.

A multidatabase system is likely to have many different representations for similar data objects.

Structural differences. Depending on how an object is used by a database, it may be structured differently in different local databases. A data item may have a single value in one database and multiple values in another. An object may be represented as a single relation in one place or as multiple relations in another. The same item may be a data value in one place, an attribute in another, and a relation in a third place.

Missing or conflicting data. Databases that model the same real-world object may have conflicts in the actual data values recorded. One system may not have some information recorded due to incomplete updates, system error, or insufficient demand to maintain such data. More serious is the case where two databases record the same data item but give it different values. The values may differ because of an error or because of valid differences in underlying semantics.

Heterogeneous local databases. Many multidatabases claim to support heterogeneous data models at the local level — generally, the network, hierarchical, and relational models. The support mainly consists of providing local translation capability from the local model to the common global model, typically relational. Even systems that only support relational DBMSs may be heterogeneous to some degree. The relational data model is a theoretical model and different implementations may interpret the theoretical model differently.

Supporting local DBMS heterogeneity requires a trade-off between writing translation code and limiting participation. If the multidatabase developers are willing to write enough translation code (considering development costs and execution efficiency), the multidatabase can accept a wide variety of local DBMSs. Another consideration here is that global system software must program any local functional deficiencies. If minimizing translation code cost is important, then the variety of DBMSs allowed to join the multidatabase will be limited to those with interfaces close to the global standard.

Global constraints. Because different local databases may represent semantically equivalent data or semantically related data, the global system needs some method for specifying and enforcing integrity constraints on interdatabase dependencies and relationships (global constraints). These constraints may represent additional semantic information about the data items involved.

Global integrity constraints are sometimes defined as part of the global schema. Other multidatabases keep separate auxiliary databases specifying global constraints. The query processor checks these auxiliary databases during query execution to enforce the constraints.

Global constraints require a thorough system policy statement that defines how to manage them. An example is an interdatabase update dependency, where updating an object in one database should update an equivalent object in another database. If the update is to be propagated, the site autonomy of the second node may be compromised. If the update is not to be propagated, the first site must either reject updates (restrict function) or accept them (violate the constraint and cause data inconsistency).

Global query processing. The basics of global query processing are consistent across most multidatabases. A user employs the global schema to submit a global query, or in the case of a multidatabase language, the query itself contains all the information necessary for retrieving local data. The query is decomposed into a set of subqueries — one for each local DBMS that will be involved in query execution. The query optimizer creates an access strategy that specifies which local DBMSs are to be

involved, what each will do, how to combine the intermediate results, and where global processing will occur. Then the access strategy is executed. Global constraints must be checked and enforced during query execution. Initial query processing usually occurs at the node where the query is submitted, while query execution is distributed across the system.

During global query execution, queries may be translated several times as they travel through the various system layers. Translations allow different languages and representations at different layers and also resolve representation differences.

Query decomposition and optimization in a distributed system have been studied in the distributed database environment and several solutions to these problems are available.[1] Multidatabase systems must also handle interdatabase dependencies, manage global resources, and support additional language features (for multidatabase languages). Interdatabase dependencies may cause functions to cascade to many databases beyond the immediate scope of a query. The query processor must manage global resources, such as the global constraints database, local workspaces, and software modules responsible for global processing. These resources are generally distributed across the system. Finally, multidatabase language systems provide many new language features that must be handled by the query processor. All these demands on the query processor must be handled in an efficient manner, despite its dynamic, distributed nature and the lack of control over local DBMSs.

Concurrency control. The traditional concept of a transaction as short-lived and atomic is unsuited to the multidatabase environment. Multidatabase transactions will typically involve multiple, separate local DBMSs and several layers of data/query translations.[4] More importantly, local DBMSs have site autonomy; so global control does not, in fact, include control of the actual data items. Multidatabase transactions are relatively long-lived and often nonatomic.

Concurrency control schedules concurrent-transaction data accesses to be serializable. However, this requires knowledge of all currently active transactions and the ability to control access to data items. A standard DBMS user interface does not normally provide information about other users' transactions or access to data item locks, timestamps, etc. Moreover, different DBMSs may use different local concurrency control schemes. The global system has enough information to provide concurrency control for global transactions, but it does not have information about local transactions. Therefore, it cannot provide total concurrency control. Consequently, many multidatabases, particularly global schema multidatabases, restrict global information access to retrieval only. Updates must be performed through the local DBMS interface on a node-by-node basis.

Security. Distributed-system security is difficult.[5] Some of the problems include nonsecure communication links, varying levels of security at different nodes, and the large number of global user types to support. Multidatabases currently rely on underlying hardware and system software for most of their security requirements. Site autonomy provides some measure of local security because local DBAs can restrict the information available to global users. The use of views for global users is also an important security measure. Accessing multiple systems typically means entering multiple identification and authorization codes. Within a single system, accessing multiple data items can mean acquiring multiple authorizations. The global system must manage these multiple security requirements in an automated fashion, while still respecting the intentions of the security mechanisms. Little work has been reported on the specific security requirements of the multidatabase environment.

Local node requirements. Multidatabases require global data structures and global software modules to implement global functions. Although site autonomy guarantees local DBMSs will be unchanged by joining a multidatabase, the local machine will have to share some of the global storage and processing requirements. Many multidatabases spread the load evenly over all participating sites. Others use server machines to perform most of the global functions, thus allowing the rest of the nodes to support less global function. This arrangement permits small or limited-capacity machines to participate in the multidatabase.

Global data structures and global software functions vary among multidatabase systems. Common data structures include the global schema, auxiliary databases for global constraints, space for intermediate query results, and temporary workspaces for global functions. Common software functions include translation between local and global languages, transformation between local and global information representations, query processing and optimization, and global system control.

Design approaches

There are two major approaches to designing a multidatabase system: global schemas and multidatabase languages. The global schema approach was used first in multidatabase design and continues to be a popular choice in many projects. The multidatabase language approach was inspired partly by the problems inherent in the global schema approach and partly by the simpler overall system architecture.

Global schema. The global schema approach to multidatabases is a direct outgrowth of distributed databases. The global schema is just another layer, above the local external schemas, that provides additional data independence. Consequently, some of the work in distributed-database research is applicable, particularly in the area of global schema design. A major difference, however, is that the global system cannot force local systems to conform to any sort of standard schema design (local schemas are developed independently), nor can it control changes to the local schemas. Another major difference is that a multidatabase global schema may integrate local schemas from multiple data models.

The global schema approach can make global access quite user friendly. Global users essentially see a single, albeit large, integrated database. The global interface is independent of all the heterogeneity in local DBMSs and data representations. Because most global schema multidatabases employ the relational model, users get a familiar and intuitive paradigm for accessing the system. For specific users and applications, views may be defined on top of the global schema to tailor the interface. The global schema is usually repli-

cated at each node for efficient user access.

Global schema design. Global schema design takes the independently developed local schemas, resolves semantic and syntactic differences among them, and creates an integrated summary of all their information. This is also referred to as view integration. Because of representation differences and interdependencies between data at different nodes, this process is more difficult than just taking a union of the input schemas.

There are several common techniques for integrating multiple, distinct schemas.[6,7] Similarities and conflicts between objects and relationships in separate schemas must be analyzed before they can be integrated. Some methodologies use special data models and design languages with special constructs to resolve representation differences. Generalization hierarchies are often used to classify similar objects from different schemas.[8] Finally, there may be hundreds or thousands of schemas to integrate, and the size of the task complicates the design process.

Despite the methodologies, algorithms, and heuristics that help automate parts of the schema integration process, this process is still very human-labor intensive. Global DBAs must design the global schema. These designers must have extensive knowledge of all the input schemas and global requirements to decide how to integrate the inputs (that is, to decide which of the many possible global schemas to create). An optimal design for global requirements will likely conflict with some local optimizations, but the global DBA has no control over autonomous local nodes. Therefore, the global DBA must also understand all the local optimizations and consider them when trying to create efficient global structures. The amount of global knowledge required is a major problem with the global schema approach.

Global schema maintenance. A global schema can be a very large data object, thus making it difficult or impossible to replicate at nodes with limited storage facilities. The popularity of personal computers and small DBMSs that might want to join the multidatabase system makes this an important problem. Some systems get around this by only replicating the global schema at specified server nodes. However, this means queries cannot be processed at all query origin nodes.

Global DBAs must also maintain the global schema in the face of arbitrary changes to local schemas. The literature is largely silent on how to do this. Changes to local schemas must be reflected in the global schema. The integration techniques used in global schema design and the types of changes in local data representations can complicate the mapping of changes to the global schema. Local changes can force the DBA to reconsider many design decisions made during the initial integration process — with wide-reaching consequences. Again the DBA must have extensive global knowledge of all the input schemas, the global schema, and the initial design decisions.

Language features

The features described here are taken from MDSL, the multidatabase language for MRDSM (Multics Relational Data Store Multiple).[10] A key aspect of MDSL is that query constructs remain valid while the dynamic environment changes (that is, the environment comprising the open local databases). At different points in the query execution, the operation may be applied to a single object, to multiple objects in different databases, or to an empty set (all pertinent databases are closed). Results of the operation should be consistent and intuitive in all cases.

The ability to define aliases and abbreviations for data item names is important for resolving name differences between databases. A name should be allowed to refer to multiple objects from different sources if the user thinks the objects are semantically equivalent. Thus, an operation on a named object may actually cause multiple operations to occur.

A user query might have to create temporary structures to hold intermediate results or new representations of information. A dynamic attribute is a temporary attribute defined by a mapping from existing attributes. Dynamic attributes can be used to accomplish transformations in data format, to abstract attribute values from multiple sources into a single set of values, and to create a column for joins with other relations.

Multidatabase language. The multidatabase language approach is an attempt to resolve some of the problems associated with global schemas, such as up-front knowledge required of DBAs, development time to create the global schema, significant maintenance requirements, and processing/storage requirements on local nodes. A multidatabase language system puts most of the integration responsibility on users, but eases the burden of their tasks by giving them many support functions and by providing more control over the information. Examples of such functions are included in the sidebar on language features. Most multidatabase languages are relational, similar to SQL in the standard capabilities, but extend the function significantly. Witold Litwin has argued persuasively for multidatabase languages and performed significant research in this area.[9,10]

Many of the language extensions beyond standard database capabilities are involved with manipulating data representations. Since representation differences exist when the user submits a query, the language must be capable of transforming source information into the representations most useful to the user. It is particularly desirable in this context to make the language nonprocedural. The multidatabase system should be capable of making some implicit decisions in interpreting what the user wants to accomplish and providing many functions by default.

Multidatabase language system users must have a means to display what information is available from various sources. The user is assumed to have well-defined ideas about what information is required and where it probably resides. Otherwise, the magnitude of the information available globally will make finding necessary data an overwhelming task. The language should provide the ability to limit the scope of a query to the pertinent local databases.

In summary, the multidatabase language approach shifts the burden of integration from global DBAs to users and local DBAs. Users must have some global knowledge of representation dif-

Table 2. Global schema multidatabase projects.

System Name/Organization	Stage of Development	Global Data Model	Global Updates?	System Emphases or Key Features
ADDS(Amoco Distributed Database System)[1] Amoco Research Center	Limited prototype	Extended relational	In near future	Comprehensive function, powerful user interface, global constraints
Dataplex[2] General Motors Research	Limited prototype	Relational	Yes	Query decomposition and optimization
DQS (Distributed Query System)[3] CRAI, Italy	Prototype	Relational	No	Query optimization
EDDS (Experimental Distributed Database System)[4] University of Ulster	Prototype	Relational	Yes	Small machines can join system
HD-DBMS (Heterogeneous Distributed-DBMS)[5] UCLA	Research	Entity-relationship		Global access path information, external views in multiple data models
JDDBS (Japanese Distributed Database System)[6] Japan Information Processing Development Center	Limited prototype	Relational	Yes	Based on a broadcast network
Mermaid[7] Unisys	Prototype	Relational		Query optimization
Multibase[8] Computer Corporation of America	Limited prototype	Functional	No	Comprehensive function
MultiStar[9] Consortium headed by CRAI, Italy	Prototype	Relational	No	Query processing, ability to link to other multidatabases
NDMS (Network Data Management System)[10] CRAI, Italy	Prototype	Relational	Yes	Query optimization
Preci*[11] University of Aberdeen	Limited prototype	Relational	Yes	Replicated data, nodes can support different levels of global function
Proteus[12] British universities	Prototype	Abstracted conceptual	No	Star network topology, multiple global access languages
Scoop[13] Universities of Paris and Turin	Research	Entity-relationship		Study mapping algorithms between system levels
Sirius-Delta[14] INRIA, France	Limited prototype	Relational		Translations to/from pivot system (global data model/language)
Unibase[15] Institute for Scientific, Technical, and Economic Information, Poland	Research	Relational		Global constraints
XNDM (Experimental Network Data Manager)[16] National Bureau of Standards	Limited prototype	Relational	Yes	Data translations, use of server nodes for global processing

ferences and data sources, but only about the information actually used. Multidatabase language systems trade a level of data independence (the global schema hides duplication, heterogeneity, and location information) for a more dynamic system and greater control over system information.

Review of existing projects

Tables 2 through 5 review most of the current multidatabase projects reported in the literature. These projects come from a variety of countries and institutions. Some are mainly research vehicles to study specific problem areas; others are full commercial systems. The range of organizations and the number of projects indicate the importance of this field. The tables compare high-level details and include a primary reference for each project. The re-

References

1. Y. Breitbart, P.L. Olson, and G.R. Thompson, "Database Integration in a Distributed Database System," *Proc. Second Int'l Conf. Data Eng.*, IEEE Computer Society Press, Los Alamitos, Calif., Order No. 655 (microfiche), 1986, pp. 301-310.
2. C.W. Chung, "Dataplex: An Access to Heterogeneous Distributed Databases," *Comm. ACM*, Vol. 33, No. 1, Jan. 1990, pp. 70-80 (with corrigendum in *Comm. ACM*, Vol. 33, No. 4, p. 459).
3. V. Belcastro et al., "An Overview of the Distributed Query System DQS," *Proc. Int'l Conf. Extending Database Technology*, Springer-Verlag, 1988, pp. 170-189.
4. D.A. Bell, J.B. Grimson, and D.H.O. Ling, "EDDS — A System to Harmonize Access to Heterogeneous Databases on Distributed Micros and Mainframes," *Information and Software Technology*, Vol. 29, No. 7, 1987, pp. 362-370.
5. A.F. Cardenas, "Heterogeneous Distributed Database Management: The HD-DBMS," *Proc. IEEE*, Vol. 75, No. 5, May 1987, pp. 588-600.
6. M. Takizawa, "Heterogeneous Distributed Database System: JDDBS," *Data Eng.*, Vol. 6, No. 1, 1983, pp. 58-62.
7. M. Templeton et al., "Mermaid — A Front End to Distributed Heterogeneous Databases," *Proc. IEEE*, Vol. 75, No. 5, May 1987, pp. 695-708.
8. T. Landers and R.L. Rosenberg, "An Overview of Multibase," *Proc. Second Int'l Symp. Distributed Databases*, North-Holland, Amsterdam, 1982, pp. 153-183.
9. D. Bell et al., "MultiStar: A Multidatabase System for Health Information Systems," *Proc. Seventh Int'l Congress European Federation for Medical Informatics*, Vol. 1, Edizione Luigi Pozzi, 1987, pp. 564-568.
10. W. Staniszkis et al., "Architecture of the Network Data Management System," *Proc. Third Int'l Seminar on Distributed Data Sharing Systems*, North-Holland, Amsterdam, 1985, pp. 57-75.
11. S.M. Deen, R.R. Amin, and M.C. Taylor, "Implementation of a Prototype for Preci*," *Computer J.*, Vol. 30, No. 2, 1987, pp. 157-162.
12. P.M. Strocker et al., "Proteus: A Heterogeneous Distributed Database Project," Cambridge Univ. Press, 1984, pp. 125-150.
13. S. Spaccapietra et al., "Scoop — A System for Cooperation Between Existing Heterogeneous Distributed Databases and Programs," *Database Eng.*, Vol. 5, No. 4, 1982, pp. 288-293.
14. C. Esculier, "The Sirius-Delta Architecture: A Framework for Cooperating Database Systems," *Computer Networks*, Vol. 8, No. 1, 1984, pp. 43-48.
15. Z. Brzezinski et al., "Unibase — An Integrated Access to Databases," *Proc. 10th Int'l Conf. Very Large Databases*, Morgan Kaufmann, San Mateo, Calif., 1984, pp. 388-396.
16. S.R. Kimbleton, "Applications and Protocols," *Distributed Systems — Architecture and Implementation*, Springer-Verlag, 1981, pp. 308-370.

ported functional content for each project is based on the reference material.

Blank table entries mean the information was not apparent from the reference. "In the near future" indicates that a function has been designed but its implementation is not yet complete. Several stages of project development are indicated:

- Research — a functional system has not yet been implemented;
- Limited prototype — the system has been implemented, but is not fully functional;
- Prototype — a full system has been implemented within the organization; and
- Commercial — the system has been implemented and is available for purchase.

The column for system emphases and key features presents important or unusual features of the system. It does not give a comprehensive picture of available system functions.

The majority of systems under study are global schema multidatabases. The close ties to distributed databases allow some synergy in solving related issues in the two fields. The single data-source user paradigm is also appealing. However, problems of size and complexity make global schema multidatabases impractical for large distributed systems. Since the trend today is toward more interconnection (that is, toward larger systems), multidatabase language systems — the other major design approach — seem more practical for many future requirements.

Multidatabase language systems have no constraints on system size, but the trade-off for achieving this is a multiple data-source user paradigm and a more complex user interface. The access language has more features, and users must have more knowledge about data locations and local representations. A testimony to the effectiveness of multidatabase language systems relative to global schema multidatabases is the fact that homogeneous multidatabase language systems are the first class of multidatabases to produce commercial products, albeit with limited functions.

There are a number of open problems to solve to make multidatabase systems more useful and efficient. First, global system requirements should not be equally distributed among all nodes. Today's distributed systems contain a variety of machine types and sizes, and small machines should be able to join the system with minimal local overhead required. Some existing multidatabase systems recognize this need.

In addition, multidatabase systems should make better use of powerful server nodes to perform the bulk of global processing. The large processing requirements and heterogeneous nature of multidatabase systems seem ideally suited to incorporating specialized data-

Table 3. Federated database projects.

System Name/Organization	Stage of Development	Global Data Model	Global Updates?	System Emphases or Key Features
Heimbigner[1] University of Colorado	Limited prototype	Object oriented	Yes	Language support for data transformations, negotiation protocol for input schema creation
Ingres/Star[2] Relational Technology, Inc.	Commercial	Relational	In near future	Can define multiple import schemas at a node
Superdatabases[3] Columbia University	Research		Yes	Hierarchical system structure, concurrency control

References

1. D. Heimbigner and D. McLeod, "A Federated Architecture for Information Management," *ACM Trans. Office Information Systems*, Vol. 3, No. 3, 1985, pp. 253-278.
2. W. Litwin and A. Zeroual, "Advances in Multidatabase Systems," *Proc. European Teleinformatics Conf. — Euteco 88, Research into Networks and Distributed Applications*, North-Holland, Amsterdam, 1988, pp. 1,137-1,151.
3. C. Pu, "Superdatabases: Transactions Across Database Boundaries," *Database Eng.*, Vol. 10, No. 3, 1987, pp. 143-149.

Table 4. Multidatabase language system projects.

System Name/Organization	Stage of Development	Global Data Model	Global Updates?	System Emphases or Key Features
Calida[1] GTE Research Labs	Prototype	Relational	Yes	Query optimization, implicit joins
Hetero[2] Felipe Carino, California	Prototype	Extended relational	Yes	Powerful user interface
Linda[3] Technical Research Center of Finland	Limited prototype	Relational		Close to being an interoperable system rather than a multidatabase
MRDSM (Multics Relational Data Store Multiple)[4] INRIA, France	Prototype	Relational	Yes	Comprehensive function, many language features, global constraints
Odu[5] University of Wales	Limited prototype	Entity-relationship	No	Small machines can join systems
SWIFT (Society for Worldwide Interbank Financial Telecommunication)[6] SWIFT, Europe	Research	Relational		Transaction structure and processing
VIP-MDBS (Vienna Integrated Prolog-Multidatabase System)[7] Vienna Technical University	Prototype	Relational		Global language is Prolog

References

1. W. Litwin and A. Zeroual, "Advances in Multidatabase Systems," *Proc. European Teleinformatics Conference — Euteco 88, Research into Networks and Distributed Applications*, North-Holland, Amsterdam, 1988, pp. 1,137-1,151.
2. F. Carino Jr., "Hetero: Heterogeneous DBMS Frontend," *Office Systems: Methods and Tools*, North-Holland, Amsterdam, 1987, pp. 159-172.
3. A. Wolski, "Linda: A System for Loosely Integrated Databases," *Proc. Fifth Int'l Conf. Data Engineering*, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 1915, 1989, pp. 66-73.
4. W. Litwin, "An Overview of the Multidatabase System MRDSM," *Proc. ACM Annual Conf.*, ACM Press, New York, 1985, pp. 524-533.
5. A.O. Omololu, N.J. Fiddian, and W.A. Gray, "Confederated Database Management Systems," *Proc. Seventh British Nat'l Conf. Databases*, Cambridge Univ. Press, 1989, pp. 51-70.
6. B. Holtkamp, "Preserving Autonomy in a Heterogeneous Multidatabase System," *Proc. 12th Int'l Computer Software Applications Conf. — Compsac 88*, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 873, 1988, pp. 259-266.
7. E. Kuhn and T. Ludwig, "VIP-MDBS: A Logic Multidatabase System, *Proc. First Int'l Symp. Databases in Parallel and Distributed Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 893, 1988, pp. 190-201.

Table 5. Homogeneous multidatabase language system projects.

System Name/Organization	Stage of Development	Global Data Model	Global Updates?	System Emphases or Key Features
Empress[1] Rhodius Inc.	Commercial	Relational	Yes	Limited global language function
Sybase[1] Sybase, Inc.	Commercial	Relational		Limited global language function
System R*[2] IBM	Prototype	Relational	Yes	Query optimization

References

1. W. Litwin and A. Zeroual, "Advances in Multidatabase Systems," *Proc. European Teleinformatics Conference — Euteco 88, Research into Networks and Distributed Applications*, North-Holland, Amsterdam, 1988, pp. 1,137-1,151.
2. B.G. Lindsay, "A Retrospective of R*: A Distributed Database Management System," *Proc. IEEE*, Vol. 75, No. 5, May 1987, pp. 668-673.

base machines.[11] Adding a database machine to the underlying distributed system could dramatically enhance performance if the multidatabase system could dynamically redistribute global processing to make use of the added capability.

Another problem is the lack of support for identifying specific data in a system, given the possible huge amounts of data available globally. Multidatabase systems provide location independence for data, but only after the user has precisely identified the data to access. Discovering the precise identifiers for desired data can be a problem if the user does not already know them. Most existing multidatabase systems provide users with the ability to scan the available data sequentially, but this approach is impractical for large systems. If a user cannot easily find the data that he or she wants to access, then the system has failed to provide global information access. Multidatabase systems should provide some navigational aids to help the user find the desired data. One possible approach is to provide a summarized view of the global information available.[12] A summary data representation would be more compact, therefore easier to search. Another approach would be to add intelligence to the system so it could identify specific data when given imprecise identifiers.[12]

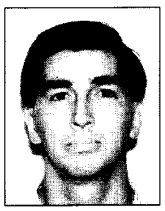
Related to the problem of searching large amounts of data is the problem of identifying semantically equivalent data in different local databases. Because of the large amounts of data involved, searching for all semantically identical entities can be a lengthy process. Most existing multidatabase systems provide functions for mapping entities to a common data representation after semantic equivalence has been established, but identifying semantic equivalence is an open problem. Again, a summary representation of system data could help by giving semantically equivalent entities a common representation at a more abstract level.[12] ■

Acknowledgments

This work has been partially supported through the IBM Resident Study program.

References

1. S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, New York, 1984.
2. H. Garcia-Molina and B. Kogan, "Node Autonomy in Distributed Systems," *Proc. Int'l Symp. Databases in Parallel and Distributed Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 893, 1988, pp. 158-166.
3. L. DeMichiel, "Performing Operations over Mismatched Domains," *Proc. Fifth Int'l Conf. Data Engineering*, CS Press, Los Alamitos, Calif., Order No. 1915, 1989, pp. 36-45.
4. V.D. Gligor and R. Popescu-Zeletin, "Concurrency Control Issues in Distributed Heterogeneous Database Management Systems," *Proc. Third Int'l Seminar on Distributed Data Sharing Systems*, North-Holland, Amsterdam, 1985, pp. 43-46.
5. C. Wang and D.L. Spooner, "Access Control in a Heterogeneous Distributed Database Management System," *Proc. Sixth Symp. Reliability in Distributed Software and Database Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., Order No. 737 (microfiche only), 1987, pp. 84-92.
6. C. Batini, M. Lenzerini, and S.B. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, Vol. 18, No. 4, Dec. 1986, pp. 322-364.
7. A. Motro, "Superviews: Virtual Integration of Multiple Databases," *IEEE Trans. Software Eng.*, Vol. 13, No. 7, July 1987, pp. 785-798.
8. U. Dayal, "Processing Queries over Generalization Hierarchies in a Multidatabase System," *Proc. Ninth Int'l Conf. Very Large Databases*, Morgan Kaufmann Publishers, San Mateo, Calif., 1983, pp. 342-353.
9. W. Litwin, "From Database Systems to Multidatabase Systems: Why and How," *Proc. Sixth British Nat'l Conf. Databases*, W.A. Gray, ed., British Computer Soc., Cambridge Univ. Press, 1988, pp. 161-188.
10. W. Litwin and A. Abdellatif, "An Overview of the Multidatabase Manipulation Language MDSL," *Proc. IEEE*, Vol. 75, No. 5, May 1987, pp. 621-632.
11. A.R. Hurson et al., "Parallel Architectures for Database Systems," *Advances in Computers*, Vol. 28, M.C. Yovits, ed., Academic Press, San Diego, Calif., 1989, pp. 108-151.
12. M.W. Bright and A.R. Hurson, "Summary Schemas in Multidatabase Systems," Computer Eng. Tech. Report TR-90-076, Pennsylvania State Univ., University Park, Pa., 1990.



M.W. Bright is a PhD student in the computer engineering program at Pennsylvania State University. He is supported through IBM's Resident Study program. His research interests include distributed systems, multidatabases, linguistics, system integration, and computer security.

A.R. Hurson is a computer engineering faculty member at Pennsylvania State University. His research for the past 12 years has been directed toward the design and analysis of general-purpose as well as special-purpose computer architectures. Hurson has published more than 100 technical papers. He served as co-guest editor of a special issue of the *IEEE Proceedings on Supercomputer Technology* and is coauthor of an IEEE Computer Society Press tutorial on parallel architectures for database systems.

Hurson is a member of the IEEE Computer Society Press Editorial Board and an IEEE distinguished visitor.

Simin H. Pakzad is an assistant professor of computer engineering at Pennsylvania State University. Her research interests for the past six years include parallel processing, interconnection networks, database systems, and fault-tolerant multicomputers. Pakzad has published more than 40 technical papers in the areas of databases, database machines, and fault-tolerant interconnection networks.

Pakzad received her MS in computer science from the University of Iowa and her PhD in computer science from the University of Oklahoma.

Readers may contact A.R. Hurson at 121 Electrical Engineering East, Pennsylvania State Univ., University Park, PA 16802; e-mail a2h@ecl.psu.edu.