VLDB Journal, 2, 285–310 (1992) Dennis McLeod, Editor

Federated Databases and Systems: Part II – A Tutorial on Their Resource Consolidation

David K. Hsiao

Received July 11, 1990; revised version received June 13, 1991; accepted August 22, 1991.

Abstract. The issues and solutions for the interoperability of a class of heterogeneous databases and their database systems are expounded in two parts. Part I presented the data-sharing issues in federated databases and systems (Hsiao, 1992). The present article explores resource-consolidation issues. *Interoperability* in this context refers to data sharing among heterogeneous databases, and to resource consolidation of computer hardware, system software, and support personnel. *Resource consolidation* requires the presence of a database system architecture which supports the heterogeneous system software, thereby eliminating the need for various computer hardware and support personnel. The class of heterogeneous databases and database systems expounded herein is termed *federated*, meaning that they are joined in order to meet certain organizational requirements and because they require their respective application specificities, integrity constraints, and security requirements to be upheld. Federated databases and systems are new. While there are no technological solutions, there has been considerable research towards their development. This tutorial is aimed at exposing the need for such solutions. A taxonomy is introduced in our review of existing research undertakings and exploratory developments. With this taxonomy, we contrast and compare various approaches to federating databases and systems.

Key Words: Interoperability of heterogeneous databases and systems (attribute-based, hierarchical, network, relational, object-oriented); data-sharing techniques (database conversion, schema transformation, transaction translation, data-model-and-language-to-data-model-and-language mappings).

1. Introduction

A new class of heterogeneous databases and systems consisting of both new and old existing systems is currently being developed. Until now, each of these databases and

---

David K. Hsiao, Ph.D., is Professor of Computer Science, Naval Postgraduate School, Monterey, CA. (Reprint requests to Prof. Hsiao, Code CS/Hq, Bldg, Ha-221, Naval Postgraduate School, Monterey, CA 93943.)

Editorial note: Portions of the introductory sections of Part II of this article replicate material in Part I (Hsiao, 1992) in order to provide the reader with a self-standing article.

systems has been used in a stand-alone environment for a specific application with a particular integrity constraint and unique security requirement. They have been rather efficient in their respective applications and effective in upholding their respective integrity constraints and security requirements. However, these databases and systems are heterogeneous. In order to facilitate their respective application, integrity, and security requirements, they must have their respective data models, data languages, and database systems. They are also supported on different computer hardware, system software, and professional personnel, since all the stand-alone environments are distinct. What makes them into a class are:

1. they all belong to an organization which requires data in various heterogeneous databases to be shared so that new applications dealing with organizational matters can be developed from the shared data, i.e., heterogeneous data meant to be shared in the organization;

2. they, on the other hand, have their own individual application specificities and must uphold the individual integrity constraints and security requirements, i.e., data-sharing among heterogeneous databases meant to be controlled by individual database systems;

3. these heterogeneous databases and systems of the organization must be consolidated if the data sharing and its access controls are to be effective and efficient (i.e., controlled sharing of heterogeneous databases meant to be facilitated in a single architecture instead of many separate and stand-alone environments).

Such a class of heterogeneous databases and systems is termed *federated databases and systems.*

In this section we reveal the need for federated databases and systems. We report the factors that have prompted the pending arrival of federated databases and systems in Section 1.2. In Section 1.3, we list the requirements for heterogeneous databases and systems to be federated. However, the bulk of the tutorial deals with various approaches towards a solution to controlled sharing of heterogeneous data (Section 2) and various architectures for consolidating heterogeneous database-system resources. To this end, we propose a taxonomy for the purpose of examining various approaches and architectures in terms of the requirements set forth in Section 1.3.

With this taxonomy, we are able to compare the merits and demerits of various approaches and architectures. It is important to note that heterogeneous databases

and systems in a federation consist of both new, old, and older databases and systems. This is a real-world database problem which we attempt to address. Thus, our sample data models and data languages include new ones, such as the object-oriented; old ones, such as the relational; and older ones, such as the hierarchical and the network. It is also important to note that approaches and architectures examined herein are mostly research proposals and exploratory developments. The technology for federated databases and systems is simply not here now. We hope this tutorial will prompt more researchers and explorers into the field of federated databases and systems in the coming decades.

Some colleagues working in the area of federated databases and systems term this area the *interoperability of heterogeneous databases and systems.* Others term it the *interoperability of multidatabase systems.* Either terminology is acceptable to us. However, some prefer the term *integration* over the term interoperability. We take issue with the use of the term of integration, because integration means combining databases and systems into a whole. In federated databases and systems, there are approaches and architectures which require no combination of multiple databases and systems into one. We may examine the integration issues of, for example, concurrency control mechanisms of various heterogeneous databases systems and database schemas of various heterogeneous databases at some "lower" level. Integration will be used in addressing these lower-level issues in the tutorial, but it will not be used to refer to the new area of study known as federated databases and systems.

1.1 What Is the Notion of Federated Databases and Systems? In a federation of heterogeneous databases, there is the need for data sharing among the diverse databases, and for resource consolidation of all supporting software, hardware, and personnel, although each database has its own autonomy in terms of, for example, its integrity constraint, application specificity, and security requirements. Thus, federated databases and systems deal with heterogeneous databases. They must provide data sharing and resource consolidation without violating the autonomies of individual databases. In the following, we first report the proliferation of the heterogeneous databases in an organization. We then propose controlled sharing of data among heterogeneous databases. The sharing is not only necessary, but controlled, so that the autonomy of each database is upheld. Lastly, we explore the need for resource consolidation.

1.2 What Factors Prompted the Pending Arrival of Federated Databases and Systems? There are several real-world factors which are elaborated in the following

sections, although database technologists and researchers may not be aware of their concerted impact on the need for and arrival of federated databases and systems at the moment.

1.2.1 The Replacement of Traditional Data Processing Practices with Modern Homogeneous Database Systems. In traditional data processing, data are stored on tapes. A data processing task requires the manual handling of tapes and transactions. To process data on tapes for a new query, data processing professionals must first write the necessary transactions off-line and then run the new transactions against necessary tapes manually. For routine data processing involving regular and standard transactions, there is already considerable manual work which is not only error-prone, but also labor-intensive. For ad hoc queries, their routines and practices are greatly affected. There are more errors in developing the new transactions and in running them against existing tapes. There are also more labor-intensive efforts on the part of data processing professionals in developing new transactions and in handling tapes. The introduction of modern database systems as replacements for traditional data processing practices has greatly reduced labor-intensive and error-prone pitfalls. Since data are stored on disks and managed by the database system automatically, there is no manual handling of storage media. Regular and standard transactions are cataloged in and executed by the database system routinely. There is also no need for any manual handling of regular and standard transactions. For ad hoc queries, the modern database system provides an ad hoc query capability. Thus, database professionals may develop new transactions for queries on-line and query the existing database directly. Finally, each modern database system is highly specialized to deliver the most effective and efficient on-line processing of a class of data processing tasks. For example, the relational database system is particularly suitable for keeping records. Thus, data processing tasks on payroll records, on employee records, and on other record collections may be taken over by the relational database system. Despite the diversity of the record-keeping tasks and differences in the record type, the same relational database system can handle them effectively and efficiently. Thus, the relational database system is said to handle homogeneous databases of records, since all the records are stored on the disks with the same format (i.e., the relational form), and are accessible and manipulatable by transactions written in the same relational data language (e.g., SQL). The sameness (i.e., homogeneity) in the data model and data language, is introduced, of course, for the effective and efficient handling and processing of the intended databases. For these reasons, the relational database system is termed a homogeneous database system for record keeping.

There are other homogeneous database systems which are specialized in other distinct and major tasks beyond data processing of records. For example, the hierarchical database system supports the hierarchical databases; the network database system the network databases; the functional database system the functional databases; the object-oriented database system the object-oriented databases; and so on. The great proliferation of many homogeneous databases and database systems indicates that traditional data processing (using tapes and relying on manual handling of tapes and transactions) is disappearing. It also indicates that the homogeneous database systems not only replace traditional data processing tasks, but also open up new database applications. Thus, database systems become an indispensable means in an organization for handling information needs.

1.2.2 The Proliferation of Heterogeneous Databases in an Organization. Typically, each department in an organization has its own information needs and focuses on a specific database application. For example, the personnel department may use a record-keeping database system to keep track of employee records. The use of a relational database system such as ORACLE to support the database, and writing transactions in SQL to access and manipulate employee records, has been in vogue.

The engineering department may focus on design specifications in terms of product assemblies. Each assembly consists of many subassemblies, each subassembly many components, each component many parts, each part many design specs, each spec many figures and numbers. These design specs can best be organized as a hierarchical database of facts and figures supported by a hierarchical database system. Thus, we may use, for example, an IBM IMS to support the hierarchical database and a data language, DL/I, to write transactions for accessing and manipulating the database.

The inventory department, on the other hand, may wish to use a network database to represent the many-to-many relationships among their inventory records. For example, a part (therefore, a part record) may be supplied by (related to) several suppliers (several supplier records); a supplier (a supplier record) on the other hand may be supplying (related to) several parts (part records). Thus, in this example, there is a many-to-many relationship between the part records and the supplier records. There are many such many-to-many relationships in a real inventory collection (i.e., a network database). Such databases can be supported by a network database system such as Unisys DBMS 1100 and by transactions written in a network data language such as CODASYL-DML.

The research-and-development department may want to experiment with a functional database to support expert-system applications using a functional database system such as CCA Local Database Manager and a data language such as CCA Daplex. It may also desire to try an object-oriented database system and its object-oriented data language on new applications in manufacture engineering. Many new and experimental object-oriented database systems and data languages (e.g., HP IRIS) have been proposed and prototyped recently.

Databases, data languages, and database systems in different departments, although homogeneous with respect to their own departments, are *heterogeneous* in the organization; this is because they are based on different data models, data languages, and database systems. If departments of an organization attempt to computerize all their useful information into databases, using suitable database systems and employing stylized data languages to write transactions for their highly specialized applications, then it is inevitable that a proliferation of heterogeneous databases and systems will result. As we enter the Information Age, the race towards computerized information and the proliferation of heterogeneous databases, data languages, and database systems in an organization will be intensified. This proliferation is not reversible; nor can the proliferation be restricted to one data model, to one data language, and to one database system. In other words, the proliferation is on the heterogeneity of databases and systems in all the departments, not just on a collection of homogeneous databases and systems in a single department.

1.2.3 Data Sharing of Various Databases in the Organization. The effective use of information scattered in different departments requires data sharing among the departments for corporate planning and decision-making, marketing strategies, regulatory compliances, inter-departmental communications and coordinations, and other multi-departmental activities. In fact, the effectiveness of sharing data within an organization may well be the most important surviving factor of the organization in the Information Age. What would be the most expeditious way to share data among heterogeneous databases? There are three requirements in federated databases and systems:

1. The first requirement is that the user must be able to access a heterogeneous database as if it were the user's homogeneous database. In other words, the user should not be required to learn the data model of the heterogeneous database. Nor should the user be required to write transactions in the data language supported by the other database system of the heterogeneous database. Instead, the user continues to view the heterogeneous database

by way of the user's familiar data model and writes transactions against the database in the user's familiar data language. For example, a relational database user in the personnel department may access a hierachical database in the engineering department as if it were a relational database by writing SQL transactions for such accesses and manipulations of the database. We term this requirement the *transparent access to heterogeneous databases.*

2. The second requirement allows the owner of a database to share the database with others without compromising the owner's integrity constraint, application specificity, and security requirement. In other words, the autonomy of the owner's database is upheld, despite the fact that multiple accesses and manipulations are being made by users of other departments. We term this requirement the *local autonomy of each heterogeneous database.*

3. The third requirement is that federated databases and systems are multimodel and multilingual. By *multimodel* we mean that a database system in the federation supports various databases in many different data models. For example, a multimodel database system may support relational databases, hierachical databases, network databases, functional databases, object-oriented databases, and other model-based databases. By *multilingual* we mean that the database system executes transactions, each of which may be written in a distinct data language, for its corresponding model-oriented databases. For example, a multilingual database system may execute SQL transactions against relational databases, DL/I transactions against hierachical databases, CODASYL-DML transactions against network databases, Daplex transactions against functional databases, and transactions written in an object-oriented or new data language against object-oriented or new databases, respectively. Without being multimodel and multilingual, federated databases and systems will not be able to support heterogeneous databases and systems which are the necessary condition of the federation.

Unless the aforementioned three requirements are met, data sharing among heterogeneous databases scattered in different parts of an organization (i.e., federation) will not become effective. Here, the emphasis of requirements is on the effectiveness of federated databases and systems.

1.2.4 Resource Consolidation of Supporting Software, Hardware, and Personnel. Heterogeneous databases scattered in different departments in an organization are likely to be supported respectively by different sets of computer hardware,

database systems, and database professionals. Such supports are both inefficient and unaccountable. They are inefficient due to the duplication of hardware, software, and personnel in supporting several, separate, and complete database systems and their databases. They are unaccountable because if there is any difficulty in data sharing it is hard to hold a particular department and its database system responsible for the difficulty. Consequently, communications and cooperations among the departments in terms of data sharing will be hindered. The question is whether or not it is possible to come up with an architecture for federated databases and systems so that inefficiency and unaccountability of heterogeneous databases for data sharing will be resolved. Later we will review several architectures for federated databases and systems. First we will spell out requirements for such an architecture.

The architecture of federated databases and systems must be special-purpose and parallel. This requirement may overcome inefficiency and unaccountability issues. By *special-purpose* we mean that the computer and its secondary storage are dedicated to and specialized in the support of the databases and database-system software. Due to the recent advances in computer technology, it is entirely cost-effective to construct special-purpose computers for better database management performance than mainframes and superminis. These special-purpose database computers are termed database backends, or, for short, *backends.* The backend architecture must also be parallel. Parallel backend architecture is termed the *multibackend* architecture. Specialization and parallelism are the two most important architectural principles for the improvement of the computer performance and efficiency.

By *multibackend* we mean that federated database systems, whether centralized or distributed, have been off-loaded from the mainframe computers into specialized backend computers. They can be supported by a single backend and its database store. However, they are likely run on multiple backends and their respective database stores where the backends, not database stores, are interconnected by way of a communication net. With identical backends, this architecture requires that the database-system software be *replicatable* over the identical backends. However, the federated databases are not replicated. They are required, nevertheless, to be *clustered* or *partitioned.* The distribution of data aggregates in a cluster must induce parallel accesses to all the aggregates in the cluster. Thus, the distribution and redistribution of federated databases on existing and new database stores are required to be *automatic.* When the number of the backends at a site is two or greater, the backends and their stores are configured parallel to sustain the *multiple-transactions-and-multiple-database-streams* (MTMD) operation. These requirements allow federated databases and systems to be run more efficiently with built-in, processing-and-accessing parallelism, to be maintained by fewer personnel,

to be supported with identical hardware, replicatable software, and reconfigurable databases, and to be charged with the sole responsibility for the support of federated databases and the database-system software.

It is important to note that, unlike the previous requirements for data sharing which emphasize the effectiveness of federated databases and systems, these architectural requirements emphasize efficiency of the federated databases and systems. The architecture of the multibackend database system allows the user to scale the system in terms of the number of backends and their stores, i.e., the degree of its parallelism, for the performance gain and capacity growth of federated databases and systems.

For *accountability,* we require federated database systems to provide deadlock-free accesses to their databases, although these accesses may have already met integrity constraints, application specificities, and security requirements. Otherwise, concurrent accesses for authorized and necessary data will be indefinitely delayed or deferred. Thus, the search for effective and efficient access and concurrency controls in federated databases and systems is aimed to address the accountability issue. We also can discuss in the following section the need for effective and efficient access and concurrency controls in terms of their necessary role in upholding the local autonomy of a federated database system and its databases.

1.2.5 Access and Concurrency Controls for Local Autonomies of Federated Databases. To uphold the local autonomy of departmental databases in terms of integrity constraint, application specificity, and security requirement, accesses to departmental databases must be controlled. Thus, federated databases and systems are required to provide *access and concurrency control mechanisms* for triggering the particular integrity constraint, for interfacing with the specific model/language software, for enforcing the necessary security requirement, and for controlling concurrent accesses to heterogeneous databases of separate departments. The question is, therefore, what would be the most effective and efficient architecture for the incorporation of necessary access and concurrency control mechanisms into federated databases and systems. These architectural issues are addressed later in this paper.

Here, we simply point out that effective and efficient *access and concurrency controls* are necessary for upholding autonomies of local databases and thus a requirement for federated databases and systems to be truly effective in data sharing and highly efficient in resource consolidation. Consequently, this requirement underscores all previous requirements. In our examination of various approaches towards either data sharing or resource consolidation we must examine these approaches for their capacity to incorporate necessary access and concurrency control mechanisms.

1.3 Summary of Five Requirements for Federated Databases and Systems. All solutions for and approaches to federated databases and systems presented in this article will be examined in terms of five requirements. Data sharing requires:

1. Transparent accesses to heterogeneous databases in the federation;
2. The local autonomy of each heterogeneous database;
3. Multimodel and multilingual capabilities of federated database systems.

Resource consolidation requires:

4. Multibackend capability.

Upholding local autonomies of federated databases requires:

5. Effective and efficient access and concurrency control mechanisms.

Since approaches to data sharing have been expounded upon in Part I (Hsiao, 1992), we will now explore solutions for resource consolidation.

2. Architectures for Resource Consolidation

In Sections 1.2.4 and 1.2.5, requirements for resource consolidation were established to address the issues of efficiency and accountability of federated databases and systems. In order to be efficient, federated databases require architectures which will induce

1. parallel operations (i.e., the multiple-transactions-and-multiple-database-streams [MTMD] operation with special-purpose and multibackend computers),
2. identical backends and their database stores,
3. replicatable database-system software on backends,
4. clustered or partitioned databases on database stores,
5. automatic database distribution or redistribution on a per-cluster or per-partition basis over parallel stores.

For accountability, the architecture should be able to facilitate effective and efficient access and concurrency controls over the federated databases

1. without causing an indefinite delay in any access,

2. without breaching the constraints, specificities, and requirements for any transaction.

In following sections, we examine three entirely different architectures of federated databases and systems for resource consolidation. They are mainframe-based, single-backend-based, and multiple-backend-based. In examining each of them, we refer to the requirements of efficiency and accountability.

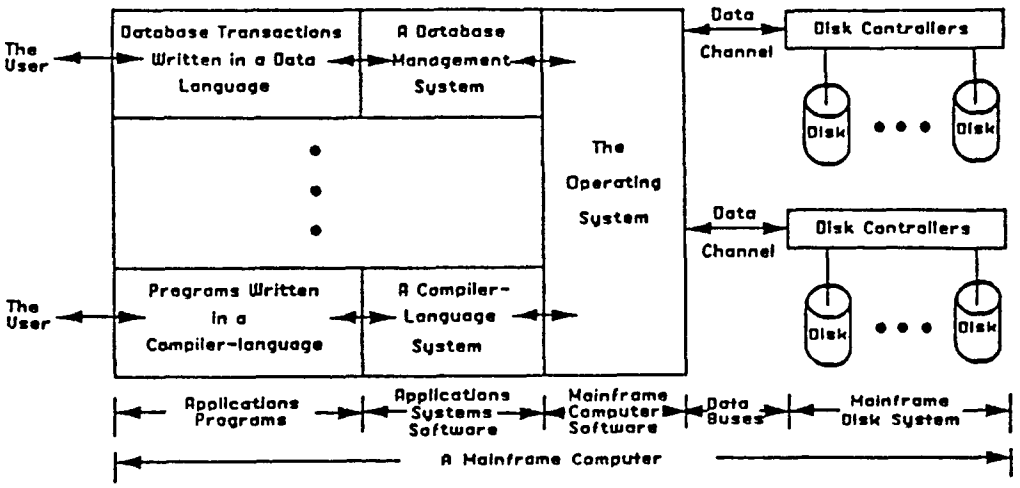
2.1 Sharing a Mainframe Computer among Federated Databases and Systems

In a mainframe computer (e.g., a supercomputer or a supermini), database transactions are executed by the mainframe as *application programs* (Figure 1). There are many kinds of application programs in a mainframe computer. Each kind is supported by an *application system.* The mainframe computer and its operating system therefore run many application systems.

2.1.1 Merits in Consolidating Database Transactions and Database Systems in a Mainframe. Each federated database system is considered an application system by the mainframe. Transactions of a database system in the federation are considered as application programs of the database (or more precisely, application) system of the mainframe. Since there are many heterogeneous database systems in the federation, there are many application systems supported by the mainframe and its operating system. This is a good case for resource consolidation of the database-system software and transactions into a single mainframe.

There are, of course, other nondatabase application programs and systems. Consider compiler-language systems. The mainframe typically runs several compiler-language systems as application systems. In this kind of application system, we have program-language applications. Programs written by the user and compiled by the compiler-language system are also termed application programs of the mainframe computer. Thus, as the application-system software, heterogeneous database systems must share resources of the mainframe not only with other application-system software (such as compiler-language systems), but also among heterogeneous database systems themselves.

Figure 1. Mainframe-Based Approach to Federated Database Systems



2.1.2 Difficulties in Consolidating Federated Databases and Systems in a Mainframe. Sharing mainframe-computer resources among application-system software, particularly heterogeneous database systems, presents difficulties:

1. The disks of the mainframe must be shared among application-system software. Whereas compiler-language systems utilize disks as temporary storages, which enhances storage sharing, heterogeneous database systems utilize disks for their databases as permanent storages which precludes sharing the same set of disks for each other's databases. Consequently, sharing mainframe disks with heterogeneous database systems is not possible.

2. As an application system, a compiler-language system may be CPU-bound and main-memory-bound during the program-compilation phase. Nevertheless, it is seldom I/O bound. Further, during the execution of a compiled program as an application program, the typical program does not consume a lot of computer resources. During the execution of a transaction, on the other hand, the database transaction is typically CPU-bound, memory-bound, and I/O-bound. Due entirely to its data-intensive and data-voluminous operations, a database transaction as a mainframe application ties up considerable mainframe computer resources.

3. The intensive demand on computer resources by various application-system software requires periodic upgrading of the underlying mainframe computer. The replacement of the existing mainframe computer with a newer and more powerful mainframe tends to be disruptive and costly, since we must shut down the existing computer system and bring up the new system. Further, new lines of mainframes are always more costly than the current line of the mainframe computer.

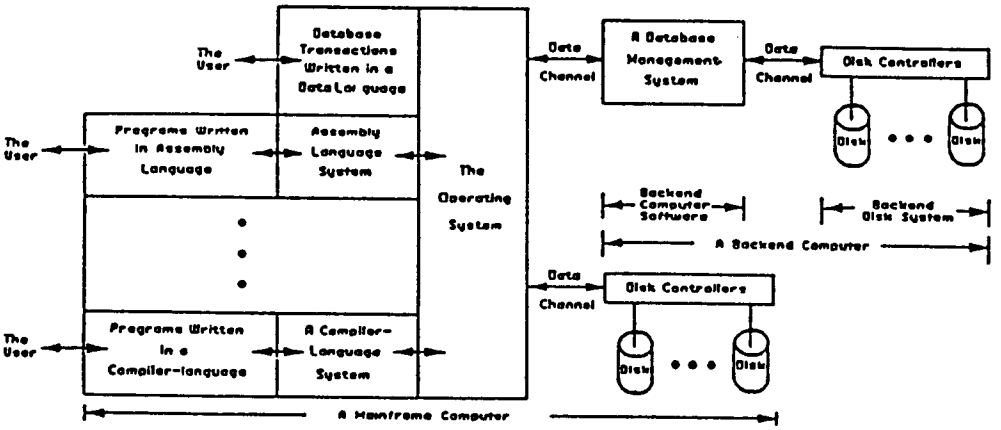
With these difficulties, heterogeneous database systems seldom share a mainframe computer with other application-system software. In fact, heterogeneous database systems do not even share the same mainframe computer among themselves. Since a mainframe is either a supercomputer or a supermini, the use of a single mainframe computer to support exclusively a monomodel/monolingual or bimodel/bilingual database system is a costly undertaking. The replacement of the mainframe with a more powerful and newer mainframe for performance enhancement and capacity growth of a single database system with limited heterogeneity is therefore disruptive and costly, too.

Since by nature federated databases and systems have greater heterogeneity, the use of a large number of mainframe computers for supporting federated database systems will not only incur greater costs and disruptions in the replacement issue, but also create difficulties in the development of parallel operations and concurrency controls. Except in the case of the database conversion approach to data sharing (Hsiao, 1992) where two database systems (e.g., relational DB2 and hierarchical IMS) and their databases are supported by a single mainframe computer, the use of multiple mainframe computers for federated databases and systems has not been in practice. Thus, we disqualify this architecture from the onset without checking it against the remaining architectural requirements.

2.2 Use of Single-Backend Computers for Federated Databases and Systems

Single backends are minis, supermicros, or database machines (single or dual processor). As Figure 2 shows, the term *backend* indicates that the computer or machine is in the *back* of either a mainframe computer or a communication device. Further, federated databases and database systems of the mainframe computer are *off-loaded* from the mainframe computer onto backend computers or machines. Symmetrically, we term the mainframe computer and the communication device *frontends*. To utilize a backend computer or machine, the user typically accesses it through a frontend, i.e., by way of a mainframe computer or via a communication device such as a local-area network. While both the mainframe and the communication

Figure 2. The Single-Backend Architecture for Federated Database Systems



device support nondatabase applications, database applications are solely supported by the single-backend computers and machines. Thus, database backends become heterogeneous database servers to the mainframe computer and communication device.

Since each backend (computer or machine) consists of its own set of disks, it may dedicate its entire resources to a heterogeneous database system and its databases in the federation. If the heterogeneity of federated databases and systems in the federation is *m*, then we may have m database servers, each of which is monomodel and monolingual and supports a collection of homogeneous databases. Together, they provide for the federation multimodel and multilingual capabilities and serve as a reservoir of heterogeneous databases.

2.2.1 Database Conversion Approach to Data Sharing. Separate, monomodel and monolingual database systems are now off-loaded from the mainframe computer into separate single-backends. Database converters (Hsiao, 1992) may still run at the mainframe or be relegated to their respective *source* backends. For example, the hierarchical-to-relational database converter may be relegated to the hierarchical database backend (i.e., the source backend), which would convert a hierarchical database into a relational database and send the relational copy to the intended relational backend, the *target* backend, by way of the frontend. With the use of multiple single-backends for the heterogeneous databases in the federation, the

routing of conversion tasks and converted copies still rests with the frontend. Such routing has minimal impact on the performance and load of the frontend.

2.2.2 For Single-ML-to-Single-ML, Single-ML-to-Multiple-MLs, and Multiple-MLs-to-Multiple-MLs Mappings. The multi-system architecture of these three mappings (Hsiao, 1992) can be supported by multiple single-backend computers or machines as configured for the data conversion approach in the previous paragraph. For Single-ML-to-Single-ML mapping, for example, if the heterogeneity of the federation is three (e.g., relational, hierarchical, and network) then we can have three database servers (i.e., the relational/SQL server, the hierarchical/DL/I server, and the network/CODASYL-DML server) being realized in three different backends. For this mapping, their schema transformers and transaction translators can run in the frontend or in their respective source backends. More specifically, the relational-to-hierarchical schema transformation and the SQL-to-DL/I transaction translation can be done either in the frontend or in the relational/SQL server but need not in both, for instances. Essentially, in this case either the frontend or the backend transforms the schema, translates the transaction, and ships the transformed schema and translated transaction to the target backend for execution. The six source-to-target transformations and translations in this example can be supported either entirely in the frontend or two in each backend.

For Single-ML-to-Multiple-MLs mapping, the global-to-local schema transformation and global-to-local transaction translation can be done in the frontend. There is no need for a separate server for the global data model and language, since there is no real global database in the federation. However, different sets of global transformers and translators may be run on different target backends where the set on the backend produces the transformed schema (in the local data model) and the translated transaction (in the local data language) for the local database system. In this distribution of the transformers and translators, there is no need to run any set of transformer and translator in the frontend.

For Multiple-MLs-to-Multiple-MLs mapping, the intermediate ML does not have a database. Therefore, there is no need to set aside a backend for the intermediate ML. All the source-ML-to-target-ML transformations and translations by way of the intermediate ML can be done in the frontend centrally or in respective source backends separately.

It is important to observe that despite the differences in the software complexity of their transformation and translation requirements, all the three mappings discussed in this section require the use of multiple single-backend computers as in the case of database conversion approach to data sharing.

2.2.3 Multiple-MLs-to-Single-ML Mapping. We need only one single-backend computer or machine to support the single (kernel) database system and its data model and language. The multiple-MLs transformation and translation software can be supported on the frontend. Thus, the single backend becomes the kernel database server of the federation. All the heterogeneous and federated databases are stored in the kernel database server in the kernel-data-model form. There is no other database server. Since this mapping is a single-(kernel)-system architecture, it is natural to consolidate the system software into a single backend.

2.2.4 Merits and Limitations of Single-Backend Architecture. Since there are seven architectural requirements for resource consolidation and access and concurrency controls, we examine each of aforementioned architectures in terms of these requirements.

Specialization and Parallel Operations. The use of single-backend computers or machines to support federated databases has the following advantages. The cost is much lower than the cost of a mainframe computer. Further, the backend can be tailored to a given data model and data language and can dedicate its entire resources to the given database system, whereas the mainframe computer tends to host many application systems. While running several heterogeneous databases and systems on a single mainframe computer is difficult, if not impossible, running multiple heterogeneous database systems on multiple mainframe computers will incur prohibitive costs. However, running multiple heterogeneous database systems on multiple single-backend computers (one for each backend) is straightforward (e.g., there is the commercial IDM 500 for the relational databases and system, the proposal of CASSM for the hierarchical databases and system, and the experimental XDBMS for the network databases and system. These are cost-effective, single-backend, special-purpose database computers or machines).

To support parallel MTMD operations it is necessary to use a number of single-backends. Since these are separate servers hosting different databases, it is necessary to have a supervisor for such a parallel, multi-computer operation to achieve an MTMD operation over a given set of heterogeneous databases on several database servers. Neither the mainframe nor the separate servers have such built-in supervisory software. On the other hand, to achieve an MTMD operation over a given database on a single server is impossible, because the server is a single-backend. Thus, the use of multiple single-backends for a server requires a built-in supervisor as in any multi-computer case. The coordination and scheduling of heterogeneous, separate database systems on many single-backend computers for parallel operations are unsolved issues.

Identical Hardware, Replicatable Software, and Automatic Database Distribution. Identical single-backend computers such as the off-the-shelf supermicros and minis may be used in lieu of the special-purpose database machines. Since parallel operations are not achievable in a given single-backend computer and the use of multiple single-backends has not taken place, the need for replicatable software on multiple single-backends, for clustering (partitioning) a database, and for an intelligent distribution of the clustered database for subsequent parallel operations is no longer there.

Access and Concurrency Controls and Local Autonomies. Except in the Multiple-MLs-to-Single-ML mapping where a single-backend computer is used, other mappings and the database conversion approach all require multiple single-backend computers. In these multi-computer, federated databases and systems there is a lack of a global access and concurrency control mechanism. Instead, there are a large number of individual, local access, and concurrency control mechanisms in their corresponding single-backend computers. A local autonomy may be upheld by an individual local access and concurrency control mechanism. However, it is not clear if a collection of independent and separate local mechanisms can provide concurrent transactions with deadlock-free accesses to data scattered in many heterogeneous databases. In other words, whether or not we can achieve global accesses and concurrency controls with just a set of local access and concurrency control mechanisms and without a global mechanism remains a research issue.

In the case of the Multiple-MLs-to-Single-ML mapping, we can rely on the only local access and concurrency control mechanism of the kernel database system. All the access and concurrency control requirements are translated into the specifications in the kernel data model and in kernel data language and enforced by the kernel database system. Since all the heterogeneous databases are stored in the kernel-data-model form, the design and instrumentation of the kernel access and concurrency control mechanism are no different from the design and instrumentation of the access and concurrency control mechanism for either centralized or distributed homogeneous database system. There is no issue in access and concurrency controls for this kind of federated databases and systems.

2.3 Sharing a Multiple-Backend Computer for the entire Federation

In a multiple-backend computer, there is a controller computer and one or more backend computers as shown in Figure 3. However, to incur high performance and great capacity the number of backends should be two or more. In any case,

the number of controller computers remains one or zero. In the latter case, the controller-computer software runs in the frontend computer. The multiple-backend computer architecture is also termed the multibackend architecture. This is not the same as the multiple single-backend computers discussed in the previous section.

In our examination of this architecture, we first review the role of the controller computer and its relationship with the frontend and backends. We then, instead of reviewing the multiple-backend architecture for the support of various mappings and conversion approach, review the multiple-backend architecture in terms of the seven architectural requirements for the Multiple-MLs-to-Single-ML mapping. Thus, we relate the architecture with the most promising mapping for the support of federated databases at the onset.

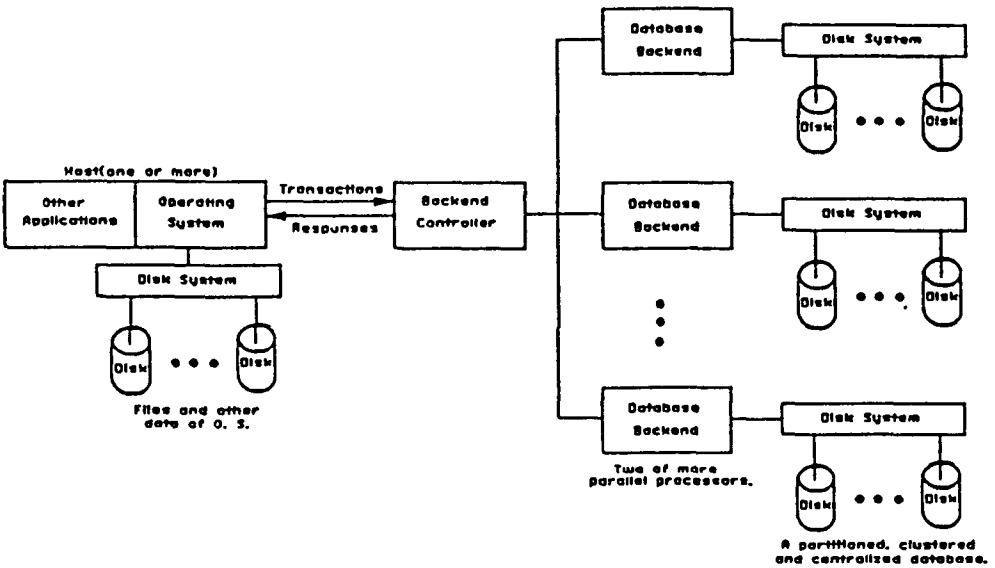
2.3.1 The Role of the Controller Computer. The controller computer merely routes transactions to backends, assembles results, and relays results to the frontend. Because the controller computer does only routing, assembly, and relay, it is not a database computer. Consequently, in some multiple-backend computers, such as the Teradata DBC 1012, the role of the controller computer is taken over by the frontend computer.

2.3.2. The Need for a Broadcasting Net. The controller computer must have an intercomputer net to communicate with backends. Minimally, the net must be able to broadcast transactions from the controller to backends simultaneously, to broadcast data from one backend to all the other backends simultaneously, and to relay messages from one computer (comtroller or backend) to another computer (backend or controller).

Some nets also do certain processing tasks such as sorting and merging results before routing them to the frontend via the controller computer. These additional capabilities are not necessary, since the individual backends can perform their respective sorting tasks and the controller can perform the merging of sorted data. It is important to note that there is no direct-coupling of backends during transaction executions and database accesses. The coupling of backends' CPUs in a close and direct fashion is not conducive to parallel operations, and is therefore absent from the multiple-backend computer.

2.3.3 Multiple-Backend Computers. The multiple-backend computer architecture is the only database computer architecture that meets the seven requirements for resource consolidation previously outlined. We review the architecture against these requirements below.

Figure 3. Multiple-Backend Computer for Federated Database Systems



Specialization, Parallel Operations, Identical Hardware, Replicatable Software, Clustered Databases, and Automatic Database Distribution. For this section, we consider six of the seven requirements, which are interrelated in the architecture. They can best be reviewed together for multiple-backend architecture. The seventh requirement will be discussed in the section below.

Unlike multiple single-backend computers, which are run independently from each other, multiple backends in a multiple-backend computer (i.e., the multibackend architecture) run identical database-system software in parallel. Thus, the system software is replicable. The parallel operation is achieved with the aid of clustering (partitioning). Although each backend computer in the multiple-backend computer has its own system of disks, a database in the multiple-backend computer is clustered where each cluster is *evenly distributed* across the disk systems of all the backend computers. With this distribution, parallel accesses of a cluster by multiple backend computers to their respective portions of the cluster are therefore facilitated. Further, transactions are broadcast by the controller to multiple backends via the network. Thus, it is possible for all the backends to execute the same transaction when it is broadcasted. Parallel executions of a transaction for parallel accesses to each cluster of a database become typical operations of the multiple-backend computer. Thus,

the STMD operation is achieved. If no data for a transaction are on a backend's disks, the backend computer can execute the next queued transaction. Thus, the concurrent execution of multiple transactions by the multiple-backend computer is also possible. In this way, the MTMD operation is achieved.

Effective and Efficient Access and Concurrency Controls and Their Support of Multiple-MLs-to-Single-ML Mapping. The multiple-backend computer is particularly suitable for supporting federated databases. Since the kernel database system must support many heterogeneous databases and many sets of schema transformation and transaction translation, it must be run on a computer which can allow great capacity growth and high performance gain. Further, it must provide effective and efficient access and concurrency control mechanism for deadlock-free accesses to heterogeneous databases for concurrent transactions in different data languages and concurrent accesses to different databases. The issues on capacity growth and performance gain are discussed in the next section. Here, we reiterate the solution suggested earlier for issues in access and concurrency controls. Since the Multiple-MLs-to-Single-ML mapping relies on a single, kernel database system, all the access requests, concurrency requirements, and control specifications which have been translated into the kernel data model and language can now be carried out by the access and concurrency control mechanism of the kernel database system. Despite the heterogeneity created by the multitude of model/language interfaces, federated databases are supported on an operating-system-like homogeneous database system (i.e., the kernel database system.) We know how to design an effective and efficient access and concurrency control mechanism for the homogeneous database system. Thus, this architecture is particularly relevant to the mapping. The multibackend architecture, despite the multitude of backends, is a single, parallel computer with many loosely-coupled backends and a homogeneous, kernelized operating-system-like database system.

Performance Gain and Capacity Growth. The multiple-backend computer exhibits the following capabilities:

1. The number of backends employed in a configuration is proportional inversely to the response-time reduction of a data-intensive transaction. For example, if we double the number of backends in a multiple-backend configuration, we can reduce the response time of the same transaction nearly by half.

2. When the database capacity is increased, the proportional addition of backends to the multiple-backend configuration can keep the response time of the same transaction nearly unchanged. For example, suppose the response time of

a transaction of a database is 2 minutes. When the database is doubled in capacity, to keep the response time nearly unchanged (i.e., about 2 minutes), one needs only to double the number of backends for the transaction's database. This is known as the *response-time invariance* for capacity growth. We note in both response-time reductions for performance gain and response-time invariance for capacity growth, we simply add backends of the same line without using more expensive and newer replacements. Further, adding identical backends on the net does not disrupt the operation of the multiple-backend computer. Although the clustered (partitioned) databases may have to be redistributed (since we have new systems of disks), the *redistribution* is done by the backend database system automatically at each addition of one or more new backends and database stores.

2.3.4 Support of the Database Conversion Approach and Three Mappings. In each of the four software methods of data sharing (i.e., the database conversion approach and three mappings: Single-ML-to-Single-ML, Single-ML-to-Multiple-MLs, and Multiple-MLs-to-Multiple-MLs), there is the requirement for the multi-system support. On the other hand, despite the multiplicity of backends in the multiple-backend architecture, the architecture is single-system. The parallelism, scalability, clustering, distribution, and redistribution are "transparent" to the system software. Thus, each of the four software methods can run its multi-system software on the multiple-backend computer as in the case of the single-backend computer. The difference is, of course, that the multiple-backend computer is a parallel database computer where the single-backend is not. Performance gains and capacity growth can be accomplished in the multiple-backend easily, whereas they cannot be easily achieved either in a single-backend or in a system of multiple single-backends.

The use of multibackend architecture for these four software methods of data sharing may also open up the possibility that autonomous local access and concurrency control mechanisms in separate, heterogeneous database systems may now be coordinated by the multiple-backend computer, since there can be one coordinator in a single-system architecture. In a multi-system architecture, such as the one with multiple, separate single-backends, it may be difficult, if not impossible, to develop a single mechanism of the multiple computers for the coordination and scheduling of autonomous local access and concurrency control mechanisms. It may be possible, however, to develop a single mechanism in the multiple-backend computer for such purposes.

3. Concluding Remarks

In this tutorial, we have pointed out that the proliferation of heterogeneous databases and database systems is likely to continue and accelerate. This is prompted by the replacement of traditional data processing with modern database systems. It is also prompted by the introduction of new data-intensive and data-voluminous applications. As monomodel-and-monolingual database systems proliferate, it is likely that in a large organization several monomodel-and-monolingual database systems for diverse applications may be utilized for organizational information needs. The interoperability of these heterogeneous databases and database systems becomes necessary for overall data sharing and resource consolidation. It also creates the issues of access controls, concurrency controls, performance gains, capacity growth, maintenance cost, and support complexity. In spite of needs and issues, the organization must uphold the local autonomies of individual database systems and their databases. Otherwise, data sharing and resource consolidation will be met with resistance.

In the context of these problems, needs, and complexities, a number of software and hardware solutions emerges. For data sharing, we have classified and described various approaches in Part I (Hsiao, 1992). For resource consolidation, there are the mainframe, single-backend, and multiple-backend approaches. These are hardware-architectural solutions and are summarized in Table 1.

In examining these solutions, the most promising system and architecture to data sharing and resource consolidation of federated databases and systems appears to be the Multiple-MLs-to-Single-ML mapping and the multiple-backend computer architecture. The issues of all of the software approaches have been discussed in Section 6 of Part I (Hsiao, 1992). Here, we focus only on the issues of the architecural solutions.

3.1 Technology Issues

From the architectural viewpoint, it seems that the multiple-backend computer provides a most viable solution to federated databases and systems. Since this architecture consists of different components, the question is therefore whether or not the technology will concentrate its effort on the improvement of the cost and the capability of the broadcasting net, of the multiple-backend computer, and the database store.

The lack of a communication net with the built-in hardware for broadcasting is evident. To incorporate this capability into the hardware of a net, the bandwidth of

Table 1. Merits and Limitations of Three Architectures for Resource Consolidation

	Mainframe-based	Single-backend	Multiple-backend
Specialization in DB operations	l	h	h
Support of multiple DB systems	l	h	h
Parallel operations with a large number	l	m	h
Identical hardware	h	h	h
Replicatable DB-system software	l	m	h
Clustered databases	n	maybe	y
Automatic DB distribution for DB stores	n	n	y
Access/concurrency control of all DBs	n	maybe	y
Scaling multiplicity of computers for performance gain/capacity growth	n	maybe	y
Cost-effective	l	m	h
disruptions due to upgrades	h	m	l

(h = high; l = low; m = medium; n = no; y = yes)

the net will have to be increased significantly, since all of the networked backends may be doing their own broadcastings simultaneously. Simultaneous broadcastings are termed *multicastings.*

6.2 Research Issues

Research issues of the architecture are many. Multibackend database computer architecture is characterized by the use of

1. distributed, scalable, and share-nothing computer hardware components and stores,

2. replicated meta data and system software on every backend,

3. evenly-distributed and clustered base data on all of the database stores, and

4. a high-bandwidth, extensible, point-to-point, broadcast, and multicast net interconnecting all of the backends.

Due to its share-nothing and distributed nature in (1), the research issue will be on the use of a large number of inexpensive hardware components and stores, instead of a small number of expensive hardware components and stores, for the performance gain and capacity growth.

In (2), there is the research issue on the maintenance and update of meta data replicated over a large number of meta-data stores. There is also the research issue of automatically generating and updating multiple copies of the system software for a specific number of backends in a given configuration.

In (3), we search for a data model with an equivalence relation for the kernel database system. Few of the conventional data models have a built-in equivalence relation for clustering the database so modeled. None of the conventional database systems has the capability of clustering the database automatically, distributing the clustered data evenly, and re-distributing the database evenly when the computer is scaled up with more backends and their respective stores.

In (4), research is urgently needed in hardware and software for broadcasting and multicasting using a high bandwidth net such as the fiber-optical local-area-net with the 100-megabits-per-second capability. The network hardware for buffering of long messages and for interrupting backends' CPUs to receive these messages by their communication processes should also be researched. Otherwise, the messages may overwhelm the network buffers; the communication processes in a backend may not be ready for the incoming messages from the net.

7. Postscript

Extensive citations for the terms used for this two-part tutorial have been given in Part I (Hsiao, 1992), including over thirty references primarily related to software issues. In this article we include only references to architectural issues. There are many papers on either the single-backend computer or the multiple-backend computer. They can be found in Hurson et. al. (1989). For additional reading on the multibackend database computer and on performance and design methodologies for the multibackend architecture, the reader may refer to Demurjian et. al. (1986*b*,

1987). The benchmark results of a multibackend computer in terms of the response-time reduction for performance gain and the response-time invariance for capacity growth can be found (Hall, 1989; Hall et al., 1990). Database partitioning for parallel accesses to the database by the multibackend computer can be found in Hsiao (1991). The use of database computer as database servers can be found in Demurjian et. al. (1985, 1986*a*). The issue of interconnecting networks for federated databases and systems can be found in Hsiao and Kamel (1991).

Acknowledgment

The work reported here is supported by funds from NPS, NPMTC, and NRL.

References

Demurjian, S.A., Hsiao, D.K., and Marshall, R.G. The configuration analysis of a database server for office automation. *Proceedings of National ACM Conference,* ACM Press, October, 1985.

Demurjian, S.A., Hsiao, D.K., and Marshall, R.G. The architectural requirements and integration analyses of a database server for office automation. *Proceedings of 1986-IFIP WG 8.4 Working Conference on Methods and Tools for Office Systems,* IFIP Press, October, 1986*a*.

Demurjian, S.A., Hsiao, D.K., and Marshall, R.G. *Design Analysis and Performance Evaluation Methodologies for Database Computers,* Prentice-Hall, 1987.

Demurjian, S.A., Hsiao, D.K., and Menon, J. A multi-backend database system for performance gains, capacity growth and hardware update. *Proceedings of the Second International Conference on Data Engineering,* IEEE Computer Society Press, February 1986*b*.

Hall, J.E. Performance evaluations of a parallel and expandable database computer: The multibackend database computer. *Master's Thesis in Computer Science,* Monterey, CA: Naval Postgraduate School, 1989.

Hall, J.E., Hsiao, D.K., and Kamel, M. The multibackend database system (MDBS): A performance study. *Proceedings of International Conference on Databases, Parallel Architectures, and Their Applications, (Parbase-90),* Miami Beach, Florida, 1990.

Hsiao, D.K. A parallel, scalable, microprocessor-based database computer for performance gains and capacity growth. *IEEE Micro,* 0:44–60, 1991.

Hsiao, D.K. Federated databases and systems: Part I–A tutorial on their data sharing. *The International Journal for Very Large Data Bases,* 1:127–179, 1992.

Hsiao, D.K. The impact of the interconnecting network on parallel database computers. *Proceedings of the Fifth International Workshop on Database Machines,* Tokyo, 1987.

Hsiao, D.K. and Kamel, M.N. The multimodel and multilingual approach to interoperability of multidatabase systems. *International Conference on Interoperability of Multidatabase Systems,* Kyoto, Japan, 1991.

Hurson, A.R., Miller, L.L., and Pakzad, S.H., eds. *Tutorial: Parallel Architectures for Database Systems,* New York: IEEE Computer Society Press, 1989.