VLDB Journal, 1, 127-179 (1992) Dennis McLeod, Editor

Federated Databases and Systems: Part I – A Tutorial on Their Data Sharing

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 presents the data-sharing issues in federated databases and systems. Part II, which will appear in a future issue, 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.

Keywords: 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 systems had been used in a stand-alone environment for a specific application with a

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.)

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 (in Part II). 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 are already considerable manual work which are 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 of 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; object-oriented database system the object-oriented databases; and so on. The great prolifer-

ation 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 their 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 have 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 in 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 is 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 is 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 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. In Part II we review several architectures for federated databases and systems. Here, we first 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 is *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 can also discuss in the following section the need of 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 will be addressed in Part II.

Here, we simply point out that effective and efficient *access and concurrency controls* are the necessary condition for upholding autonomies of local databases and 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 the previous requirements of federated databases and systems. So, in our examination of various approaches towards either data sharing or resource consolidation we must also examine these approaches in the light of their capability 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 Part I and II will be examined in terms of five requirements. For data sharing there is the requirement of:

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.

For resource consolidation, there is the requirement of:

4. Multibackend capability.

For upholding local autonomies of federated databases, there is the requirement of:

5. Effective and efficient access and concurrency control mechanisms.

2. Approaches to Data Sharing

There are several approaches towards data sharing among federated, heterogeneous databases. In our examinations we refer to not only requirements for data sharing but also for other issues.

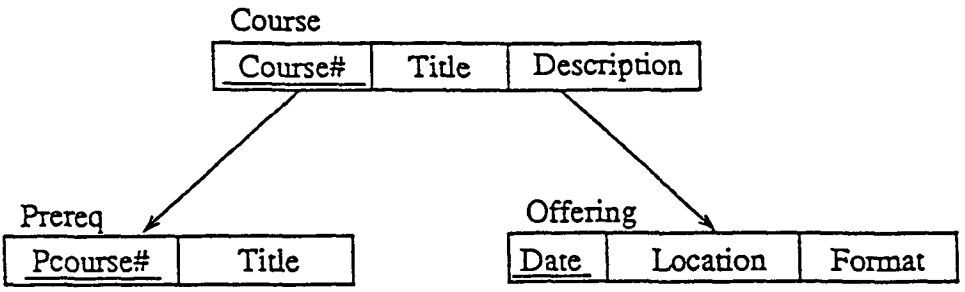
2.1 Database Conversion. To make a database available to a user who is not familiar with the data model, this approach simply converts the database into an equivalent database in the familiar data model of the user. For example, if a relational database user desires to access a hierachical database, a *database converter* may be employed

1. to access the hierachical database system for the intended database;
2. to make a copy of the hierachical database;
3. to transform the copy into an equivalent relational database;
4. to load the transformed database, i.e., the relational database, into a relational database system;
5. to allow the relational database user to access the transformed database by way of the relational database system.

In this approach, the database converter may be considered as a utility package of the operating system that supports both hierachical and relational database systems. By utilizing operating-system functions (calls), the database converter can then communicate with two different database systems for accessing one database and for storing the other database. Further, the database converter can utilize the tools, files, and other utility packages of the operating system for the conversion work.

However, the *semantic equivalence* of one database in one data model to the other transformed database in the other data model is determined by the converter. Such

Figure 1. A Hierarchical Database



Note the three *segment types:* **Course, Prereq,** and **Offering.** Each type has several *attributes* enclosed in boxes. Prereq and Offering are called *child* segment types (children). Course is called the *parent* or *root* segment type. Each type has one or more segments of the same type. A *segment* consists of *attribute values* of the segment type (not shown in this diagram).

Rules for the establishment of a hierarchical database:

1. Each parent segment may have zero or more child segments. In our example both Prereq and Offering are potential parents, even though at this point they have no child segment.

2. No two parent segments of any type or types may have a child segment in common.

These rules establish the *hierarchy* or *tree* of a database. Certain attributes of a segment type are designated as the *sequence field,* which uniquely identifies individual segments in a sequence of segments of the same type. In this sample database, **Course#, Pcourse#,** and **Date** are designated as sequence fields of their respective segments. For our convenience, they are underlined in the data diagram.

determination is never unique. What is a semantic equivalent to one user may not be a semantic equivalent to the other user, because typically the semantics of one data model may be distinctly different from the semantics of the other data model. If we want to stay in our own semantics, i.e., not to learn the semantics of the other database and its data model, we need to accept the semantic equivalence provided by the database converter on the semantics of the other database in the light of our own data model and data semantics.

In Figures 1 and 2, we provide a sample of database conversion. Figure 1 uses a data diagram to depict the logical organization of a hierarchical database. We also refer to the database and its constructs with the terminology and semantics of the hier-

Figure 2. A Relational Database

Course (Course#, Title, Description)
Prereq (Course#, PCourse#, Title)
Offering (Course#, Date, Location, Format)

The relational database is specified in the relational notation. There are three *relations:* **Course, Prereq,** and **Offering.** Each relation has several *attributes* enclosed in parentheses. There may be one or more *tuples* of attribute values in the relation. However, tuples are not shown in the parenthesized notation.

The rules of establishing a relational database are simple:

1. No two tuples of a relation have identical attribute values.
2. No tuple may have null attribute values.
3. No two relations of a database have identical attributes.

In addition, each relation may have a *primary key* consisting of one or more attributes. The attribute values of the key uniquely identify individual tuples of the relation. A relation may also have one or more *secondary keys* which uniquely identify individual tuples in other relations. For our convenience, we underline attributes that form the keys of a relation.

In this sample, two relations, Prereq and Offering, each have two attributes for their keys, whereas the relation, Course, has only one attribute for its key. Furthermore, one of the two keys of either Prereq or Offering is identical to the key of Course. These identical keys are secondary and uniquely identify tuples in the Course relation. Other keys of these relations are primary of their respective relations.

achical data model. Some explanations in generic terms are necessary, since we want to relate the database and its constructs to the transformed database in Figure 2. In Figure 2, the relational database which is equivalent to the hierarchical database in Figure 1 is presented. Here, we use the relational terminology to refer to the logical organization of the relational database and its constructs. We also use generic terms to elaborate on the semantic equivalence of the two databases and their constructs. We will not show the conversion algorithm which is the program logic of the database converter. It is hoped that by illustrating the conversion process with this simple example in Figures 1 and 2 we may be convinced that the database conversion approach to data sharing is viable. In Table 1, we summarize all the necessary notions, constructs and techniques for the conversion of a hierarchical database to an equivalent relational database.

2.1.1 Transparent Accesses to Heterogeneous Databases in a Federation. The database conversion approach to data sharing provides excellent transparent accesses

Table 1. Equivalence Table of Hierarchical and Relational Database Constructs

Hierarchical Database Constructs	Relational Database Constructs
Attribute (field name)	Attribute (attribute name)
Attribute value (field value)	Attribute value (value)
Segment	Tuple
Segment type	Relation
Sequence field	Key
Parent segment	Tuple with specific primary key value
Parent segment type	Relation with primary key
Child segment	Tuple with specific secondary key value which is identical to one and only one primary key value of the other tuple*
Child segment type	Relation with secondary key which must be identical to a primary key of the other relation
Root segment	Tuple without any secondary key value
Root segment type	Relation without any secondary key

*There are subtle considerations:

1. In Figure 2 we simply replicate the specific primary key value of the parent tuple in the child tuple of the other relation as its secondary key value. How do we know which is primary and which are secondary if there are multiple keys in a relation? In implementation we can always concatenate the secondary-key attribute with the name of the relation in which the attribute is primary. Thus the child relations become as follows: Prereq (Course.Course#, Precourse#, Title) and Offering (Course.Course#, Date, Location, Format).
2. As the hierarchy of a database gets deeper (i.e., the number of levels is large), a relation in the bottom of the hierarchy must all bear the primary keys of their parent, grandparent, great-grandparent, and any ancestral parent as secondary keys of the relation. This is necessary because we are using the keys to establish the hierarchy. On the other hand, the hierarchy is maintained with built-in pointers from one segment to other segments.
3. With parental keys, we can access bottom relations directly, whereas in the hierarchical database it is necessary to traverse the hierarchy from its root.

to heterogeneous databases in the federation. As long as all the heterogeneous databases which are *not* in the user's familiar data model have been converted into equivalent databases in the user's familiar data model, the user may view all such databases

homogeneously and access them with transactions written in the user's familiar data language. For example, in Figures 1 and 2 we have made a hierachical database available to the relational user by converting it into an equivalent relational database. Thus, the presence of a hierachical-to-relational database converter may transform all the other hierarchical databases into their relational equivalent. As far as the relational user's concern, the existing collection of databases in the relational and hierarchical data models becomes really a familiar collection of homogeneous relational databases. The relational user can continue to write transactions in a familiar relational data language, say, SQL, for the purpose of accessing the newly transformed databases. To the relational user, accesses to transformed hierarchical databases are no different from accesses to relational databases. We term such accesses *transparent* to the relational user.

What is necessary to accomplish the transparency, i.e., all databases are apparently homogeneous to this user, is the presence of many database converters—one for each unfamiliar data model of this user. Research results have indicated that it is possible to convert a database in a semantically rich data model to an equivalent database in a semantically less-rich data model. In the example on database conversion depicted in Figures 1 and 2, we consider that the semantics of the hierarchical data model is richer than the semantics of the relational data model. Thus, that example illustrates the point advanced here. On the other hand, the conversion of a database in a semantically rich data model to an equivalent database in a semantically richer data model is straightforward, since semantic constructs of the former are likely subsumed by semantic constructs of the latter. For example, a relational database may be converted into an equivalent hierarchical database where every root segment type represents a relation. Since root segment types in this example are without children, i.e., no child segment types, they do not form hierachies. They nicely characterize a collection of relations.

How is it possible then to convert a database in a semantically rich data model to an equivalent database in a semantically less-rich data model? Instead of repeating the example in Figures 1 and 2, and enumerating and elaborating various conversion processes and algorithms, we provide an analogy in data structures of programming languages. We know that these data structures are semantically richer than the data structure of the computer. Whereas data structures in programming languages deal with constructs such as named variables, lists, arrays, trees, tables, and records, the computer provides only primitive data constructs such as addressed bytes, half-words, full-words, pages and blocks. Compilers of programming languages somehow manage to convert rich data structures of programming languages into equivalent data structures of the computer, even though the computer data structure is more primitive,

i.e., semantically poor. Similarly, we can create equivalent databases in semantically poor data models for databases in semantically rich models. For instances, the object-oriented database has been converted into the equivalent relational database; the hierarchical to the relational; the relational to the attribute-based; the hierarchical to the attribute-based; the network to the attribute-based; the functional to the attribute-based; the object-oriented to the attribute-based. (See Postscripts for citations.)

2.1.2 Multimodel and Multilingual Capabilities of Federated Database Systems. Again, the data conversion approach to data sharing allows the federated databases and their database systems to include heterogeneous databases in many different data models and to support many different data languages on separate database systems for accessing their own databases. Therefore, the multiplicities of data models and data languages are a function of the number of the database converters available. With a large number of database converters, data sharing via a large number of different data languages on a large number of heterogeneous databases becomes a reality. However, most commercial multimodel and multilingual database systems are bimodel and bilingual. The limitation is caused by the following factor.

2.1.3 Local Autonomy of each Heterogeneous Database in the Federation. Database converters by nature generate multiple copies of the same database. Although the autonomy of an original database is safeguarded by the system which supports the database, the copies in different data models are now supported on different database systems. Consequently, it is difficult if not impossible to uphold the same integrity constraint, application specificity, and security requirement of the database over its copies by different systems. For example, if we update a hierarchical database via a DL/I transaction on a hierarchical database system such as IBM IMS, then the update must be carried out in the relational copy of the database by the relational database system, say, DB2. Otherwise, the data integrity of the database will be violated. Simultaneous updates by different database systems on copies of the same database are difficult to coordinate and control. Although the bimodel and bilingual capability of the IBM's hierarchical-to-relational database converter, known as the *extraction* capability, allows simultaneous updates of a database and its copy, this capability has not been extended to cover other constraints, specificities, and requirements. Nor has it been extended beyond the hierarchical-to-relational conversion.

We have had difficulties in simultaneously updating replications of the same database in a homogeneous and distributed database system such as CCA's SDD-1; the difficulties in updating simultaneously multiple copies in more than two heterogeneous forms of the same database by heterogeneous and separate database systems are surely

insurmountable. Consequently, once a copy of a database is made for the other database system, the autonomy of the database can be violated if the other database system does not safeguard the copy. Even if the other database system is willing to safeguard the integrity constraint, application specificity, or security requirement of the database, there is the difficulty of the system to carry out the enforcement on the copy, effectively.

One way to minimize the aforementioned difficulty is to ask a multimodel and multilingual professional, who understands integrity constraint, application specificity, and security requirement of the database in its native data-model form and native data-language specification, to specify for other database systems and copies equivalent constraints, specificities, and requirements in new data languages. Such a multimodel and multilingual professional is hard to come by. The viability of local autonomies of federated databases now rests with the availability of such an individual.

2.2 Database Schema and Transaction Translation. These two capabilities of a database system offer more effective approaches towards data sharing among federated databases than the database conversion approach. Let us first review these two capabilities. We then contrast their approaches with the database conversion approach.

2.2.1 The Role of Database Schemas in Federated Databases and Systems. In a database system using database schemas, there is a schema for each database in the system. A *database schema* consists of attributes, types, and other information about attribute values of the database. It also contains information about the relationship of attribute values in database aggregates such as keys, records, directories, files, and others. Although a database schema indicates the logical organization of attribute values as characterized by its data model, it does not specify the physical layout of the database. Nor does it contain attribute values themselves. It is fair to say that by looking at a database schema, we can tell the data model of the database. To this end, we refer to the database schema of a relational database as the relational database schema, for short, the relational schema. Thus, we create hierarchical schemas for hierarchical databases, network schemas for network databases, functional schemas for functional databases, and object-oriented schemas for object-oriented databases, and so on.

There are contemporary database systems which have been implemented and introduced as the schema-based database system. The CODASYL-schema-based network database system is a good example. What we are proposing is that in such a schema-based database system, we allow additional database schemas based on data

models other than the data model of the system to be created for a database. *This capability of having multiple database schemas in different data models for the same database is essential for data sharing.* Thus, instead of converting a database by making another copy, as in the previous approach to data sharing, we either convert the database schema to a new one or define a new database schema for the database. This process is called the *schema transformation* or *generation,* i.e., there is a *schema transformer* or *generator* which defines a new database schema on the basis of a new data model for the database. Database schemas allow the same database to be viewed in different data models. For example, the hierarchical database depicted in Figure 1 may have a relational schema, in additional to the hierarchical schema, so that the hierachical database may be viewed as a relational database as depicted in Figure 2. With this hierarchical-to-relational schema transformation the relational database user can now view the hierarchical database as if it is a homogeneous relational database.

Can the relational user then write the relational transactions, say, in SQL, for the purpose of accessing and manipulating of the hierarchical database? The answer is no. Viewing the database by way of a new database schema is one thing; accessing and manipulating the database via the new schema is another. In the latter case, the host database system must be able to understand the semantics of the new data language associated with the new data model of the schema. In referring to the previous example, the hierarchical database system must be able to understand SQL, in addition to its hierarchical database language, say, DL/I. In other words, the hierachical database system must also be a relational database system. Contemporary database systems are mostly *monolingual* each of which can only execute transactions written in the sole data language of that system. To overcome this important shortcoming, we introduce the concept of transaction translation in the following section.

2.2.2 The Need for Transaction Translations in Federated Databases and Systems. With the schema transformation (or generation) capability, it makes sense now to provide the transaction translation capability. By *transaction translation,* we mean that the database system which provides a new database schema can translate a transaction written in a new data language in the new data model (of the database schema) into an equivalent transaction in the data language of the database system (hosting the database). Again, using the example in the previous section, we can now assume that the hierarchical database system supports both relational and hierarchical database schemas of the same database. The relational database user can now write the relational transactions in SQL, which are then translated into the equivalent hierarchical transactions in DL/I. The translated transactions can then be easily executed in the hierarchical database system. The results in the hierarchical form can be viewed by

the relational user relationally via the relational schema.

We note that the transaction translation capability is *not* needed in the database conversion approach to data sharing, because there are multiple, complete, monomodel, and monolingual database systems in such a federation—one for each distinct pair of data model and data language of the system. It is possible for each database system with both schema transformation and transaction translation capabilities to be multimodel and multilingual.

2.3 Architectures of Database Systems with Schema Transformation and Transaction Translation Capabilities for Data Sharing. Using the schema-based database system with the transaction translation capability, there are four different ways, i.e., system architectures, to facilitate data sharing among federated, heterogeneous databases. They are termed

1. the single-model-and-language-to-single-model-and-language mapping,
2. the single-model-and-language-to-multiple-models-and-languages mapping,
3. the multiple-models-and-languages-to-single-model-and-language mapping,
4. the multiple-models-and-languages-to-multiple-models-languages mapping.

The notions and practicalities of these mappings in the light of the three requirements of data sharing are examined in following sections. (See Section 1.3 for the three requirements for data sharing.)

2.3.1 Single-Model-and-Language-to-Single-Model-and-Language Mapping. For short, this approach is termed *Single-ML-to-Single-ML mapping.* In this mapping we are concerned with the transformation of a database schema in one database model into the database schema in another data model for the same database and the translation of transactions in one data language into equivalent transactions in another data language. The mapping is used

- when one has a monomodel and monolingual database system in one department and desires to support a database application based on data in a heterogeneous database in a different monomodel and monolingual database system in another department;
- to refer to the data of the heterogeneous database by way of the user's familiar data model and access and manipulate the database via the transaction written in the user's familiar data language;

- to allow the user to view the heterogeneous database as if it is a homogeneous one, since a new schema in the user's familiar data model has been created for the database;

- to allows a user's transaction to be translated into the data language of the other database system, causing the translated transaction to be executed against the intended database there.

The results, i.e., output, of the transaction are, of course, in the form of the other data model. However, the presence of the schemas here and there for the same database allows the results to be transformed reversely from the other data-model form into the familiar data-model form of the user.

Comparing the approach here with the database conversion approach in Section 2.1, we note one major difference: in the database conversion approach multiple copies of a database are made and maintained, whereas in mapping only multiple schemas are generated and maintained. The other major difference is that new software must be developed for the conversion of a database in one data model to an equivalent database in another data model. With this mapping we need the new software instead for schema transformation and transaction translation without converting the database proper. In Figure 3, we depict two schemas—one hierarchical database schema and one relational database schema for the same hierarchical database. Comparing Figure 3 with Figures 1 and 2, we note the similarities. In addition, we depict in Figure 4 two transactions—one hierarchical database transaction in DL/I and an equivalent relational database transaction in SQL—against the same database depicted by their corresponding schemas in Figure 3.

How many schema generators and transaction translators are needed in federated databases and systems with the Single-ML-to-Single- ML mapping in order to achieve the maximum data sharing? Again, in referring to the previous example of a federation of hierachical and relational databases and systems, there are two schema generators: one for hierachical users of relational databases and one for relational users of hierarchical databases. There are also two transaction translators: one for the DL/I-to-SQL translation and the other for the SQL-to-DL/I translation. In general, we may conclude that if the heterogeneity of the federated databases is n, then there are n(n-1) schema generators: one for each heterogeneous database, i.e., one to make its databases looked like databases in other (n-1) data models and there are n such heterogeneous databases. There are also n(n-1) transaction translators: two for each pair of heterogeneous data languages.

Figure 3. Two Schemas for the Same Database

The hierarchical schema for the database depicted in Figure 1 is specified as follows:

(⟨FILE, Course⟩, ⟨Course#, k-value⟩, ⟨ Title, string⟩, ⟨Description, string⟩)
(⟨FILE, Prereq⟩, ⟨Pcourse#, k-value⟩, ⟨ Title, string⟩)
(⟨FILE, Offering⟩, ⟨Date, k-value⟩, ⟨ Location, string⟩, ⟨Format, string⟩)

The relational schema for the relational database depicted in Figure 2 is specified as follows:

(⟨FILE, Course⟩, ⟨Course#, k-value⟩, ⟨ Title, string⟩, ⟨Description, string⟩)
(⟨FILE, Prereq⟩, ⟨Course.Course#, k-value⟩, ⟨Title, string⟩)
(⟨FILE, Offering⟩, ⟨Course.Course#, k-value⟩, ⟨Date, k-value⟩,
⟨Location, string⟩, ⟨Format, string⟩)

It is important to observe that, despite the fact that these two databases are in two different models, respectively, their schema specifications are similar. In a *schema specification,* we use a pair of parentheses to enclose a schema. We use a pair of angles to enclose the attribute name and its *value type.* In addition to usual types, such as strings, integers, characters, and floating-point numbers, there is the k-value for a value of the sequence field or key. For the names of relations or segment types, the schema employs a *special attribute:* FILE. Because a database is characterized by one or more schemas, it is composed of one or more data aggregates, known to the database system as *files.*

We note that the absence of the anglized expression ⟨Course.Course#, k-value⟩ in the hierarchical database system suggests that the "key" to the parent has been facilitated physically by the pointer in the segment. On the other hand, the relational database does not use pointers. The only way to "point" to the parent is by way of a key as specified in the anglized expression.

What is the necessary number of heterogeneous, schema-based, monomodel, and monolingual database systems in federated databases and systems to achieve maximum data sharing? The answer to this question is straightforward; we follow the same conclusion, i.e., to infer the heterogeneity, n, of the federated databases in the premise as the number, m, of distinct and complete database systems in the federation. A federation of three heterogeneous databases and systems is depicted in Figure 5. In this example, the heterogeneity is three, i.e., n = 3. Consequently, there are six (i.e., 3*2) schema generators and six (3*2) transaction translators and three (m=n=3) database systems. They are annotated in Figure 5 also.

Figure 4. Two Equivalent Transactions in Two Different Data Languages

Consider the following hierarchical transaction in DL/I:

```
        GU    Course (Title = Federated Databases and Systems)
Next    GNP   Prereq
      GoTo    Next
```

Consider the following relational transaction in SQL:

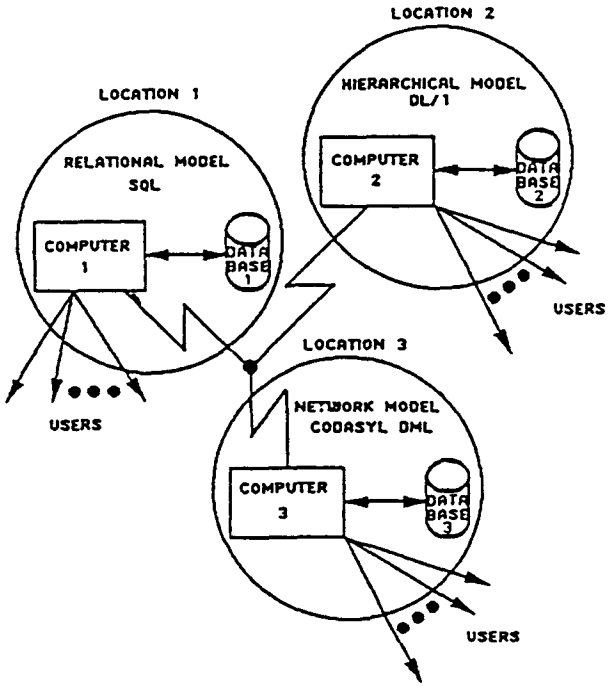
```
SELECT   Course#, Prereq.Title, Pcourse#
  FROM   Course, Prereq
 WHERE   Course.Course# = Prereq.Course#
   AND   Course.Title = Federated Databases and Systems
```

Effects of the above two transactions should be the same, i.e., they should produce an output of course numbers and titles of all the prerequisite courses for Federated Databases and Systems.

There are subtle observations: 1. In the DL/I transaction, it is clear to the reader that GNP means procuring every prerequisite course segment for the course entitled above. In the SQL transaction, it is not so clearly stated, because no hierarchy of courses is reflected by the order of statements. 2. One has to be subtle to note the difference between Prereq.Course# and Pcourse#; and between Course.Title and Prereq.Title. 3. The use of the WHERE statement, Course.Course# = Prereq.Course#, may puzzle some novice users of SQL.

We note that the Single-ML-to-Single-ML mapping provides excellent transparent accesses to heterogeneous databases in the federation. As depicted in Figure 5, each user of local databases may view heterogeneous databases in other database systems as if they are homogeneous via their built-in schemas. The user can also write transactions in the user's familiar language for the purpose of accessing and manipulating data of other databases due to the presence of the transaction translators. As long as the maximum number, n(n-1), of database schema transformers and the maximum number, n(n-1), of transaction translators are maintained in all the, n, heterogeneous database systems of the federation, no user will have any difficulty accessing and manipulating data of federated databases.

Figure 5. A Federation of Three Heterogeneous Databases and Systems



As the number, m, of heterogeneous database systems in a federation increases, the number, n, of schema generators and transaction translators must increase also. This increase may be significant, if we desire to provide maximum multimodel and multilingual capabilities for federated database systems. If we let m be n, then the multiplicative significance is in the order of n*n. The large number of schema transformers and transaction translators requires each heterogeneous database system to become an operating-system-like database system for the management of its databases, schemas, schema transformers, transactions, and transaction translators. Nevertheless, multimodel and multilingual capabilities can be realized in federated database systems with the Single-ML-to-Single-ML mapping.

Since all the accesses to a database are facilitated locally by the local database system and the shared data or databases have not been converted and copied onto the other database systems, the local autonomy can be upheld easily and effectively by the access and concurrency control mechanism of the local database system.

In this mapping, heterogeneous database schemas for the shared, local database are managed by the local database system. Also in this mapping translated transactions against the local database are executed in the local database system. Consequently, the

local database system can exercise controls over the viewing, access, and manipulation of any of its databases.

However, as federated databases and systems, the lack of an overall access and concurrency control mechanism for the federation is apparent. Thus, the remotely-entered transactions being executed locally are treated as stand-alone transactions independent from the transactions being executed in other federated database systems. The coordination and scheduling of inter-dependent transactions having been remotely entered into and executed by the various federated database systems do not exist. Consequently, the deadlock avoidance and detection may not be possible. Indefinite delays of remotely-entered transactions are also possible. As a multi-system federation, the lack of such a global mechanism may be fatal to data sharing.

2.3.2 Multiple-Data-Models-and-Languages-to-Single-Model-and- Language Mapping. For short, we term this mapping *Multiple-MLs-to-Single-ML mapping.* Assuming, for example, we have a relational database system with schema transformation and transaction translation capabilities. We can generate, in addition to relational schemas for its relational databases, hierarchical schemas for hierarchical users to access relational databases as if they are hierarchical databases, network schemas for network users to access relational databases as if they are network databases, and so on. In other words, the maximum data sharing is achieved in the relational database system by providing (n-1) schema generators: one for each non-relational database schema, and (n-1) transaction translators: one for each non-relational-to-relational translation. The complexity of federated databases and systems as opposed to the complexity of those in the previous mapping is reduced by a factor of n. More specifically, the number, m, of heterogeneous database systems in the federation is reduced to one (i.e., m=1). The number, n, of schema generators or transformers in the federation is (n-1), not m(n-1) nor n(n-1) as in other mappings. The number, n, of transaction translators in the federation is also (n-1). Since there is only one database system in the federation, all the schema transformers and transaction translators reside in the same database system, called the *kernel database system.* In fact, in this case the term, federated databases and systems, is reduced to the term, federated databases and the *system.* We note that the plural of "system" in the title of our tutorial becomes singular.

For this mapping, the impact of these two capabilities, i.e., schema transformation and transaction translation, on both conventional and modern database systems is profound. A conventional, schema-based, homogeneous database system can be modified into the kernel database system of the federation and incoporate (n-1) schema generators and (n-1) transaction translators into the system in order to provide the user with transparent accesses to heterogeneous databases in n different data models. Although

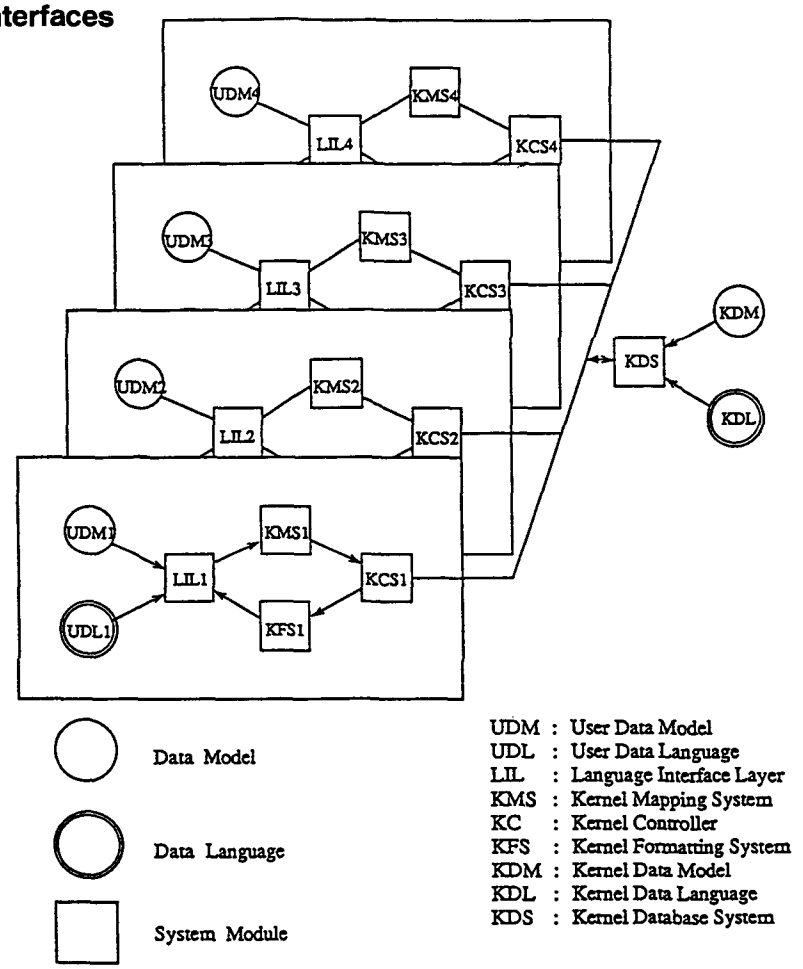
these databases are homogeneous in the data model of the kernel database system, the heterogeneity is maintained and created over the homogeneous databases of the kernel database system with the help of schemas, schema generators, transactions, and transaction translators. The merits in using this approach to data sharing are many:

- We do not have to modify the other (n-1) database systems in the federation;
- we need only to create (n-1) sets of schemas and schema transformers, instead of (n-1)(n-1) additional sets for other database systems;
- we need only to incorporate (n-1) transaction translators, instead of (n-1)(n-1) additional translators for other database systems.

The only penalty in using this approach to data sharing in a federation of conventional database systems is twofold: Existing heterogeneous databases residing on other database systems must now be reorganized into equivalent databases in the data model of the kernel database system. The reorganized databases and their associated schemas must be loaded onto the database store of the kernel database system. All the conventional database systems, except the newly modified kernel database system, must be discarded.

For the modern federated databases and system, there is nothing to be reorganized and reloaded, since databases, whether homogeneous or heterogeneous, are created by the kernel database system. The mapping eliminates multiple, separate database systems in the federation, i.e., the multi-system approach in the previous mapping; instead, it relies on a single, kernel database system, whether centralized or distributed, for the support of a collection of homogeneous databases. These databases can also be viewed, accessed, and manipulated heterogeneously, because the kernel database system provides additional schemas in other data models and translates transactions in other data languages into the kernel data language of the system for execution. The multiplicative reduction in the number of schema generators and transaction translators in this mapping makes this approach to data sharing extremely attractive. The replacement of the conventional notion of using of a number of database systems, i.e., multi-system, in a federation with the modern notion of relying on only one kernel database system in the federation, although attractive, requires new efforts. One is to make a conceptual change from the multi-system approach to federated databases to the kernel-system approach to federated databases. The other is concerned with the design and implementation of the system architecture of the kernel database system and its relationship with the schemas in different data models, schema transformers, transactions in different languages, and transaction translators.

Figure 6. A Kernel Database System with its Model/Language Interfaces



In Figure 6 we depict an experimental kernel database system which supports homogeneous databases in the attribute-based data model and executes transactions written in the attribute-based data language, ABDL. It supports heterogeneous databases by way of schemas. At the present, it supports relational databases via relational schemas, hierarchical databases via hierachical schemas, network databases via network schemas, and functional databases via functional schemas. Designs have been made to support the object-oriented databases with object-oriented schemas. Further, it executes relational transactions in SQL against relational databases with the help of the SQL-to-ABDL translator, hierarchical transactions in DL/I with the DL/I-to-ABDL translator, network transactions in CODASYL-DML with the CODASYL-DML-to-ABDL translator, and functional transactions in Daplex with the Daplex-to-

Figure 7. A Relational Database in the Attribute-Based Form

A relational database specified in the relational notation:

Course(Course#, Title, Description)
Prereq(Course.Course#, Pcourse#, Title)
Offering(Course.Course#, Date, Location, Format)

An equivalent attribute-based database specified in the attribute-based form:

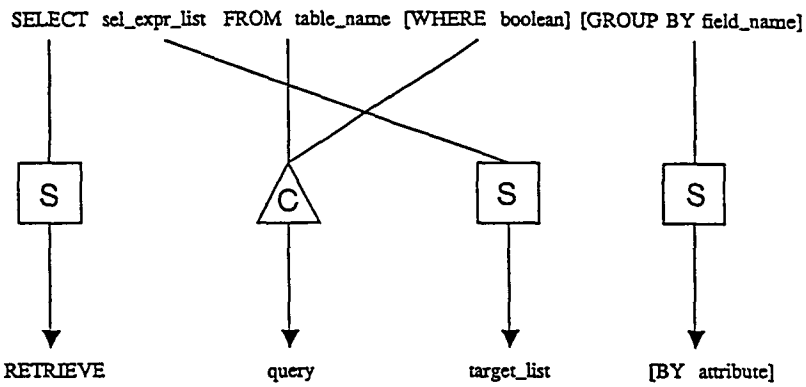
(⟨FILE, Course⟩, ⟨Course#, k-value⟩, ⟨ Title, string⟩, ⟨Description, string⟩)
(⟨FILE, Prereq⟩, ⟨Course.Course#, k-value⟩, ⟨Pcourse#, k-value⟩, ⟨Title, string⟩)
(⟨FILE, Offering⟩, ⟨Course.Course#, k-value⟩, ⟨Date, k-value⟩, ⟨Location, string⟩, ⟨Format, string⟩)

Comparing the relational database, specified in the relational notation, and its equivalent database, specified in attribute=based data model here, with the relational database in Figure 2 and its schema in Figure 3, one notes that they are identical. This is not surprising, because we use the same relational database. We use attribute-based notation to specify not only the database illustrated here, but also any attribute-based data structure, such as the schema, whether the schema is attribute-based or not. More specifically, in Figure 3 we use the attribute-based notation to specify both the relational and hierarchical schemas. Thus, these two schemas, themselves, also form two attribute-based databases.

ABDL translator. Plans have been made to provide translators for transactions written in an object-oriented data language so that they can be translated into ABDL. The pair of schema generator and transaction translator is termed the model/language interface in Figure 6. Thus, there are the relational/SQL interface, the hierarchcal/DL/I interface, the network/CODASYL-DML interface, and so on. Of course, we have the native mode, i.e., the attribute-based/ABDL interface also. This experimental federated databases and system is therefore supported by an attribute-based database system singularly, i.e., a kernel database system whose databases are in the attribute-based data model and whose transactions are written in ABDL. The heterogeneity of federated databases and transparent accesses to each heterogeneous database are made possible by the model/language interfaces.

For example, a relational user creates a relational database in the federation by using the data definition facilities of SQL to specify the database. The relational/SQL interface of the kernel database system creates an equivalent attribute-based database and its associated relational schema for the user. In Figure 7, we provide a database

Figure 8. Translating any SELECT-FROM Transaction in SQL into an Equivalent RETRIEVE Transaction in ABDL

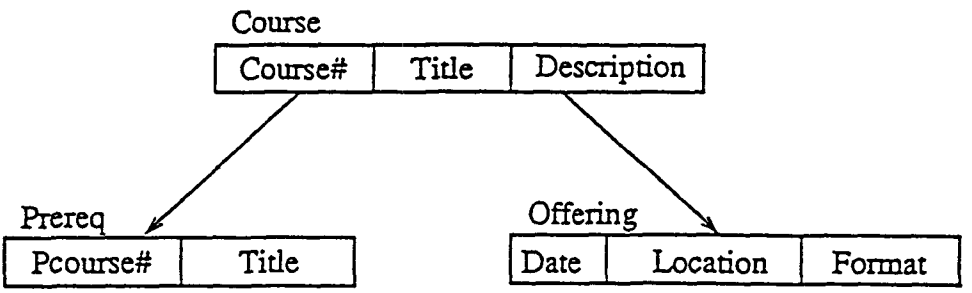


Translating any SELECT-FROM transaction with or without a WHERE or a GROUP-BY statement in SQL to the equivalent RETRIEVE transaction in ABDL involves only two translation techniques: 1. By way of simple substitutions, denoted in S. 2. Simple composition, denoted by C. Thus, SQL transactions can be readily translated into ABDL transaction in real-time.

specification in SQL of a sample relational database, a relational database schema of the sample database, and the logical organization of the database in attribute-based form, i.e., the equivalent database in the attribute-based data model. If the relational user desires to write an SQL transaction in order to access the relational database, the relational/SQL interface is invoked again. In real-time the interface retrieves the necessary relational schema, translates the SQL transaction into an equivalent ABDL transaction, and executes the translated transaction against the equivalent attribute-based database in the database store. If there are results to be routed to the relational user, the relational/SQL interface uses the same relational schema, formats results into the relational form from the attribute-based form, and routes relational results to the relational user. Some sample SQL transactions for this database and their equivalent ABDL transactions are included in Figure 8. We will not spell out algorithmic details of the schema transformation and transaction translation of the relational/SQL interface. (For more information, refer to Postscript.) As far as the relational database user is concerned, this is apparently a relational database system. Transparent accesses to federated databases have been achieved for the relational user.

Figure 9. Data-Diagram Specification of Hierarchical Database and Attribute-Based Specification of Attribute-Based Database

As in Figure 7, we borrow the same hierarchical database specified in Figure 1 and its schema specified in Figure 3 for this figure. Because the hierarchical database schema is specified in the attribute-based notation, the schema itself is in an attribute-based database. Further, it is the specification of an attribute-based database which is equivalent to the hierarchical database specified in the data diagram.



The following is an equivalent attribute-based database specified in the attribute-based form:

(⟨FILE, Course⟩, ⟨Course, k-value⟩, ⟨Title, string⟩, ⟨Description, string⟩)
(⟨FILE, Prereq⟩, ⟨Course.Course#, k-value⟩, ⟨Title,string⟩)
(⟨FILE, Offering⟩, ⟨Course.Course#, k-value⟩, ⟨Date, k-value⟩,
⟨Location, string⟩, ⟨Format, string⟩)

In Figure 9, we provide a database specification in DL/I as it is written by a hierarchical user. The hierarchical schema of the database so specified, and the logical organization of the equivalent attribute-based database are also depicted in Figure 9. In Figure 10, we illustrate the translation of hierarchical transactions in DL/I to their ABDL equivalent. It is hoped that by these illustrations the role of the hierarchical/DL/I interface is clearly outlined. Again, considerable work on the translation of DL/I transactions to their ABDL equivalent have been accomplished and implemented for the interface. (The reader may refer to Postscripts for source information.) With these illustrations, hierachical users have achieved transparent accesses to federated databases.

Figure 10. A DL/I Transaction and its Equivalent ABDL Transaction

Consider the following hierarchical transaction in DL/I:

```
        GU    Course (Title = Federated Databases and Systems)
Next    GNP   Prereq
      GoTo    Next
```

The above DL/I transaction is translated into an ABDL transaction with two retrieve statements. The *place-holder* for the output produced from the previous statement is marked with a string of asterisks. With a place-holder, the output of one statement may become the input of the next statement.

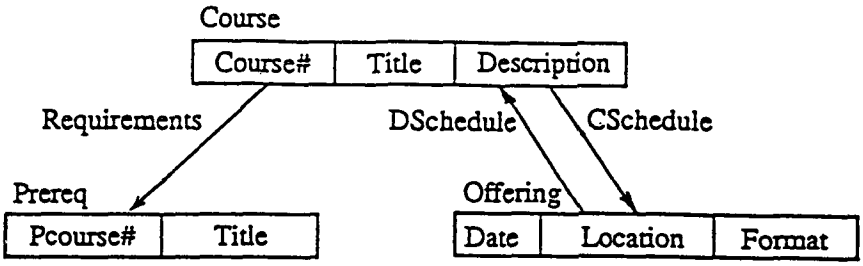
```
{RETRIEVE  ((FILE = Course) and (Title = Federated Databases & Systems))
           (Course#) BY Course#}
{RETRIEVE  ((FILE = Prereq) and (Course# = *****))
           (Pcourse#, Title) BY Pcourse#}
```

Similarly, we illustrate the working of the network/CODASYL-DML interface in Figures 11 and 12. Thus, as far as network users are concerned, their accesses to the federated databases and system are transparent. We will not illustrate the working of the functional/Daplex interface, although it has been implemented at this writing. Nevertheless, we illustrate the working of the object-oriented/OODL interface in Figures 13 and 14. Although this interface has not been implemented, the proposed design is presented herein at the suggestion of the referees of the paper. These are efforts, e.g., of supporting an object-oriented-or-complex-data-model/data-language interfaces on the attribute-based database system as their kernel.

The four previous examples of the relational, hierarchical, network or object-oriented user to create and access correspondingly a relational, hierachical, network or object-oriented database in the federation do not refer to the possibilities that, for instance, the relational user may desire to access the object-oriented database, relationally. What we have illustrated are the interfaces that produce the following four types of schemas: relational-to-attribute-based, hierachical-to-attribute-based, network-to-attribute-based, and object-oriented-to-attribute-based. We have also illustrated capabilities of interfaces to provide four types of transaction translations: the SQL-to-ABDL, the DL/I-to-ABDL, CODASYL-DML-to-ABDL, and OODL-to-ABDL. Nevertheless, we may need additional schema transformations or transaction translations

Figure 11. A Network Database and its Attribute-Based Equivalent

Following is a data-diagram specification of a network database. The many-to-many relationship between courses and offerings makes this a network database.



Following is an attribute-based specification of an attribute-based database (equivalent to the above network database).

(⟨FILE, Course⟩, ⟨DBKEY, key-value⟩, ⟨ COURSE#,string⟩,
⟨TITLE, string⟩, ⟨DESCRIPTION, string⟩,
⟨MEM.DSchedule, owner-key-value⟩,
⟨POS.DSchedule, sequence-value⟩)
(⟨FILE, Prereq⟩, ⟨DBKEY, key-value⟩, ⟨ PCOURSE#, string⟩,
⟨TITLE, string⟩,
⟨MEM.Requirements, owner-key-value⟩,
⟨POS.Requirements, sequence-value⟩)
(⟨FILE, Offering⟩, ⟨DBKEY, key value⟩, ⟨ DATE, integer⟩,
⟨LOCATION, string⟩, ⟨FORMAT, string⟩,
⟨MEM.CSchedule, owner-key-value⟩,
⟨POS.CSchedule, sequence-value⟩)

In order to capture network data constructs into the attribute-based database, a number of conventions are introduced: 1. Instead of using attributes as keys, the network database system uses a hashing algorithm to create a key for records. This system-generated key is known as DBKEY of the record type. Thus, in the attribute-based specification of the equivalent database, we make the key specification for each record type explicit. 2. Owner-and-member relationship (parent-to-child relationship in the hierarchical database) is facilitated by keys also. To distinguish from other keys, we concatenate the key attribute with MEM. *Key attributes* are names of respective relationships. 3. Included in the network data model is the *position* of a member record in a sequence of member records of an owner. Thus, in addition to key-values for DBKEY and for "pointing" to the owner, we introduce sequence-value. To distinguish the *positional attribute* from other attributes, we concatenate the membership attribute with POS.

Figure 12. A CODASYL-DML Transaction and its ABDL Equivalent

Consider the following CODSAYL-DML statements:

MOVE Cleveland TO CITY IN SA
FINDANY SA USING CITY IN SA

Underlined, capitalized words are vocabularies of CODASYL-DML. Capitals without being underlined are attributes or record types which are reflected in the network database. Attribute values are not capitalized, except in the case of character-string where the first letter may be capitalized.

The following is an equivalent ABDL transaction of the above:

{RETRIEVE ((FILE = SA) and (CITY = Cleveland)) ALL BY DBKEY}

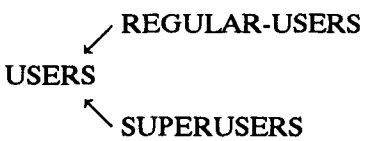
ALL is for the target list, indicating to list all the attribute values. (See Figure 8 for the syntax of ABDL's RETRIEVE statement.) the BY-clause allows records retrieved to be sorted by their DBKEY values.

or both, for instance, to allow the relational user to access an object-oriented database relationally. Let us examine the necessary addition. As it turns out the addition in the federated databases and system for this kind of mapping is minimal. The addition is restricted to the addition of schema transformers. There is no need to add more transaction translators. Let us return to the previous example where a relational database user desires to access a hierarchical database (actually, an attribute-based equivalent) relationally. At this point of the example, the hierarchical-to-attribute-based schema exists for the database. There is also two existing transaction translators - one created earlier for the relational user, i.e., the SQL-to-ABDL translator, and the other created also earlier for the hierarchical user, i.e., DL/I-to-ABDL translator. What we need now is only a hierachical-to-relational schema transformer. With the new schema transformer, we can then produce a relational schema for the intended hierarchical database. This new schema is an intermediate schema which is fed to the existing relational-to-attribute-based schema transformer. The output of the transformer is a relational schema of an attribute-based database which is equivalent to the hierarchial database intended for the relational user. The presence of the new relational-to-attribute-based schema for the hierachical database (i.e., an attribute-based equivalent) enables the relational user to write SQL transactions for the purpose of accessing and manipulating the hierarchical database (again, we recall it is an

Figure 13. An Object-Oriented Database of Users

Object class:	USERS
Attributes:	USER IDENTIFIER
	USER NAME
	USER PASSWORD
Actions:	Add-user
	Drop-user
	Change-password
Object class:	REGULAR-USERS
Attributes:	USER IDENTIFIER
	USER NAME
	USER PASSWORD
	REGULAR-USER LIST
Actions:	Add-user
	Drop-user
	Change-password
	Verify-regular-user's-password
Object class:	SUPERUSERS
Attributes:	USER IDENTIFIER
	USER NAME
	USER PASSWORD
	SUPERUSER LIST
Actions:	Add-user
	Drop-user
	Change-password
	Monitor-superuser's-access

Graphically, they are depicted in a two-level object-class hierarchy below:



We assume that named attributes in capital letters are defined in terms of primitive, built-in attributes. We also assume that the named actions (methods) are defined in terms of primitive, previously-defined actions of some given action composition rules. USERS are the *generalization* of REGULAR-USERS and SUPERUSERS, the latter a *specialization* of the former. This is the *inheritance* property of the object-oriented data model.

Figure 14. Equivalent Attribute-Based Database for an Object-Oriented Database

The following is the specification of an equivalent attribute-based database of the object-oriented database depicted in Figure 13.

(⟨FILE, USERS⟩, ⟨SKEY, key-value-for-the-generalized-object⟩, ⟨USER IDENTIFIER, key-value⟩, ⟨USER NAME, string⟩, ⟨USER PASSWORD, key-value⟩, ⟨Add-user, key-value-for-an-action ⟩, ⟨Drop-user, key-value-for-an-action⟩, ⟨ Change-password, key-value-for-an-action⟩)

(⟨FILE, REGULAR USERS⟩, ⟨GKEY, key-value-for-the-generalized-object⟩, ⟨USER IDENTIFIER, key-value⟩, ⟨USER NAME, string⟩, ⟨USER PASSWORD, key-value⟩, ⟨REGULAR-USER LIST, string⟩, ⟨ Add-user, key-value-for-an-action⟩, ⟨Drop-user, key-value-for-an-action⟩, ⟨ Change-password, key-value-for-an-action⟩, ⟨Verify-regular-user's-password, key-value-for-an-action⟩)

(⟨FILE, SUPERUSERS⟩, ⟨GKEY, key-value-for-the-generalized-object⟩, ⟨USER IDENTIFIER, key-value⟩, ⟨USER NAME, string⟩, ⟨USER PASSWORD, key-value⟩, ⟨SUPERUSER LIST, string⟩, ⟨ Add-user, key-value-for-an-action ⟩, ⟨Change-password, key-value-for-an-action⟩, ⟨ Monitor-superuser's-action, key-value-for-an-action⟩)

We observe that the notion of generalization and specialization in the object-oriented data model is realized in the attribute-based model by way of keys: GKEY for generalization and SKEY for specialization. We also observe that there are two different key values. Key values for attributes are intended for accesses to keyed attribute values. Key values for actions, on the other hand, are intended not only for accesses to keyed actions, but also for executions of the accessed actions.

Finally, we observe that there is no need to consider the translation of a transaction written in an object-oriented data language (OODL) into an equivalent transaction in the attribute-based data language (ABDL), because essential OODL constructs are embedded in the object-oriented data model as actions (methods).

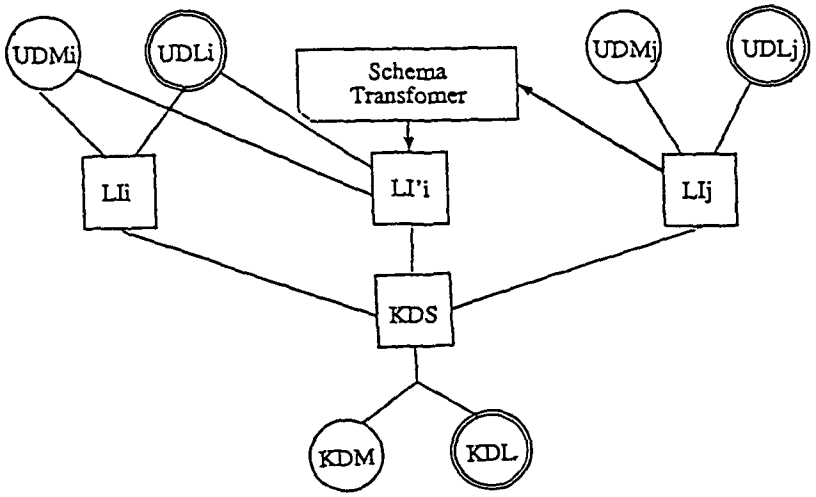
From this and previous figures, it is evident that an object-oriented/attribute-based interface may be built for the purpose of supporting object-oriented databases on the attribute-based database system as AB(Object-oriented) databases.

attribute-based equivalent in reality) as if it is a relational database. This subtle and intriguing capability is termed the *cross-model accessing capability.* In Figure 15, we illustrate the role of schemas and translators for the cross-model accessing capability in the example. Let the heterogeneity of the federated databases be n, then, in addition to (n-1) other-to-attribute-based transformers, there are (n-1)(n-2) cross-model schema transformers. Although the number of transformers have increased noticeably, the number of transaction translators remains the same, i.e., n.

The first type of transparent accesses to a heterogeneous database occurs when a user creates a database in a data model which is not the native data model of the system. For example, a relational database user desires to create a relational database in the system whereas the native data model of the system is the attribute-based data model. The system activates the relational/SQL interface which creates an equivalent attribute-based database for the relational database specified by the user in SQL. Let us call the relational database, relational, and its equivalent attribute-based database, AB(relational). The latter is, of course, heterogeneous to the relational user. However, due to the presence of the relational/SQL interface the relational user may view the AB(relational) via the relational schema created for the database as if it is a relational database. Further, the relational user may write transactions in SQL which are translated in real-time by the relational/SQL interface into equivalent transactions in ABDL which in turn access the AB(relational). Thus, as far as the relational user is concerned, the relational database is accessed by the user's SQL transactions. Transparent accesses to the heterogeneous database, i.e., AB(relational), are achieved by the presence of the interface. This type of transparent accesses allows the user to create homogeneous databases in the user's familiar data model and to access these databases with the user's familiar data language. The user is not required to learn the data model and data language of the (kernel) database system in which the database is created. Thus, the real database is heterogeneous to the user and homogeneous to the system.

The second type of transparent accesses to a heterogeneous database occurs when a user desires to share data of a database with others who are unfamiliar with the data model of the database and the data language of the user. Obviously, the database is heterogeneous to others. In order to make accesses to the heterogeneous database transparent to others, the database system activates the appropriate schema transformer and model/language interface and creates the necessary schemas for other users. For example (using the same example in the previous paragraph), if the relational user desires to share data of the user's relational database with hierarchical database users, the user may instruct the database system to provide transparent accesses to the database for hierarchical users. Since the system has cross-model capabil-

Figure 15. Cross-Model Accessing Capability



It is important to note that there are two existing model/language interfaces, LIi and LIj, in this database system. Each model/language interface consists of a schema transformer and a transaction translator. The schema transformer produces a schema of a database in the user's familiar data model, say UDLi. The database is stored in the system's kernel data model, KDM. The transaction translator translates the user's transaction from the user's familiar data language into the system's kernel data language, KDL. The execution of the user's transaction and access to the user's database are facilitated by the only database system, the kernel database system (KDS).

To allow a user to access a heterogeneous database without learning the data model of the database, the *cross-modeling capability* is provided. The model/language interface of the heterogeneous database, say, LIj, provides the schema of the database as the input to the database system. The system then produces a schema transformer which can transfer the input schema into an equivalent schema of the heterogeneous database. The new schema allows the user to view the database as if it is a homogeneous database in the user's familiar data model, UDMi. The system also borrows the transaction translator of the user's familiar data language, UDLi. The new schema transformer and the borrowed transaction translater form the new model/language interface for the user. It is denoted as LI'i.

The subscripts i and j indicate the heterogeneity of data models and languages. With the cross-model accessing capability, a user, i, can access and manipulate the database, j, with the user's familiar data model, UDMi, and language, UDLi. The "price" for the system to provide such a capability is rather small.

ity, it simply creates an additional schema for the AB(relational) database so that the same database may be viewed as an equivalent AB(hierarchical) database. It is important to note that there is only one database stored in the kernel-data-model form. The database is seen differently in different models via different schemas for the same database. Consequently, different data languages can be utilized to write transactions for accesses and manipulations of the same database by way of their respective schemas. This type of transparent accesses to heterogeneous databases in the federation allows the user to continue in the user's familiar data model and language and to access other heterogeneous databases as if they are homogeneous.

The multiplicity of data models and data languages supported in federated databases are determined by the number of model/language interfaces available in the system. They are also determined by the number and heterogeneity, n, of the heterogeneous databases in the federation. However, they are not determined by m heterogeneous database systems in the federation, since in this mapping there is only one kernel database system, centralized or distributed, which "emulates" or "simulates" heterogeneous and separate database systems.

Unlike the Single-ML-to-Single-ML mapping where the task of providing model/language interfaces, i.e., schema transformations and transaction translations, is more difficult, the task of providing model/language interfaces in this Multiple-MLs-to-Single-ML mapping is easier. This is because in the former mapping we are concerned with the schema transformation and transaction translation for each pair of m separate and different database systems. There are m(m-1) such distinct model/language interfaces at m different database systems.

On the other hand, in this mapping each pair of schema transformation and transaction translation is from a heterogeneous data model and language to the kernel data model and language. Thus, if the heterogeneity of federated databases is n, there are only n model/language interfaces and (n-1)(n-2) cross-model schema transformers in the kernel database system. Such interfaces and transformers are also easier to facilitate, since they are all mapped into the same kernel data model and data language in the same database system. Finally, they are easier to be managed and coordinated, since they are activated by the same kernel database system (and not by m multiple database systems as in the previous mapping).

Access and concurrency controls in this mapping rests with the kernel database system. All the specifications for integrity constraints, application specificities, and security requirements in any of data languages of the federation must be translated into equivalent specifications in the kernel data language of the system. Again, the translation is facilitated by the respective model/language interface. Thus, whether or not the local autonomy of a heterogeneous database can be upheld depends on

two factors: (1) Can specifications written in a data language be faithfully translated into equivalent specifications in the kernel data language of the system? (2) Will the access and concurrency control mechanisms of the kernel database system be powerful enough to carry out the intended constraints, specificities, and requirements?

The first factor is not unique to this mapping. In Section 2.3.1, we have pointed out that if a user desires to access a heterogeneous database elsewhere in the federation, the user may specify the constraint, specificity, and requirement in the user's familiar data language and submit them along with the user's transaction again written in the user's familiar data language to the other system for execution. The schema transformer and transaction translator of the other database system then translate the specification and transaction into equivalent specification and transaction in the familiar local data language of the other database system. Only transformed specifications and translated transactions are executed by the other local database system. There, the issue of "faithful" transformation and translation has been raised. If we replace the words, *other* and *local,* with the word *kernel,* we face the same issue. In either case, we have the benefit that the access and concurrency control mechanism rests with the local or kernel database system.

How can we build a most powerful set of access and concurrency control mechanisms in the kernel database system so that all the access and concurrency control specifications can be carried out by the same set of mechanisms? Here, the issue is simpler than the one in Section 2.3.1, since in that section the mechanisms, if found, must be duplicated or instrumented for all the other local database systems. Here, we need only to build one for the kernel database system. The triggering mechanism for the invocation of integrity constraints, the subschema mechanism for the specification of application specificities, the query-modification mechanism for the access control of secured data, and the concurrency control mechanism for concurrent accesses to databases can be built into the kernel database system. Access and concurrency control mechanisms of the kernel database system, whether centralized or distributed, becomes the access and concurrency control mechanism of federated databases and the system. The need of a set of global mechanisms, as in the case of the multi-system federation discussed in section 2.3.1, is not felt here. The access and concurrency control mechanism of the kernel-system of the federation is the "global" mechanism of the federation.

For instance, by extending the attribute-based data model, the schemas of the attributed-based databases, the kernel database system supports the multi-level secured databases —a requirement of the U.S. DoD. We also found that the query-modification of the kernel database system can effectively and efficiently replace the view mechanism used in the relational database system for the access control of rela-

tional databases. In other words, it is possible to translate the access control specification in SQL based on the view mechanism into the equivalent specification in ABDL based on the query-modification mechanism of the kernel database system and control subsequent accesses to AB(relational) databases. (See Postscript for further information.) Using the kernel database system to provide centralized or distributed access and concurrency controls in a federation, to emulate heterogeneous databases and systems in the federation, and to uphold their local autonomies with a set of kernelized mechanisms suggests that this is a case of strongly federated databases and their system with effective access and concurrency controls.

2.3.3 Single-Data-Model-and-Language-to-Multiple-Data-Models-and-Languages Mapping. We use the term, for short, *Single-ML-to-Multiple-MLs mapping.* With a user's familiarization of a universal data model and a universal data language, federated database systems allow the user to write transactions in the universal data language for the purpose of accessing heterogeneous databases in the federation as if they are in the universal data model. They are called universal because they are the only pair of data model and data language that provides transparent accesses to each and every heterogeneous database in the federation. For example, a relational database user in this federation may not refer to a hierarchical database relationally. Nor may the relational user to access the hierarchical database by writing a transaction in SQL. Instead, in this approach to data sharing the relational user learns the universal data model and language which allow the user to refer to the hierarchical data via the universal data model and to access the hierarchical database by writing a transaction in the universal data language.

It is also understood that none of the federated databases is in the universal data model (thus, the data model is conceptual or virtual) and none of the existing data languages resembles the universal data language (thus, the language may be more user-friendly and model-independent). The universal data model and language are provided to a user solely for the user's transparent accesses to heterogeneous databases in the federation. For this reason, they are also called the *global data model* and *global data language* of the federation. The mapping allows a user to view a heterogeneous database via the global-data-model form and not to be aware of the data model of the heterogeneous database. Accesses to the database can be facilitated by a transaction written in the global data language. The mapping translates the transaction into an equivalent transaction in the data language of the database system hosting the database. Since this is a multi-system federation, there are many different data languages. The mapping is capable of translating the same transaction in the global data language into one or more equivalent transactions in one or more data languages of

heterogeneous database systems, respectively. In other words, the user needs to learn only the universal, global data model and language for transparent accesses to all the heterogeneous databases.

The CCA's Multibase, for example, provides a new data model and language with certain simple and, yet, powerful expressions where the data model can characterize data in a number of heterogeneous databases and the data language can retrieve them for viewing, despite the fact that the user is not familiar with any of the data models and languages of heterogeneous databases in the federation. Browsing and retrieval of shareable data among heterogeneous databases are possible through Multibase. Its mapping essentially translates each transaction written in Multibase into an equivalent transaction in the data language of the heterogeneous database system which hosts the database. The mapping also transforms the data structures specified in the Multibase transaction into the data structures of the database in the data model of the database system. As far as the Multibase user is concerned, all the heterogeneous databases in various database systems of the federation are homogeneous, since they can be viewed and retrieved in the Multibase form.

The transparency of heterogeneous databases in the federation is achieved for those and only those Multibase users in the aforementioned example. In other words, if one is to enjoy the benefit of transparent accesses to heterogeneous databases, one must learn the data model and data language of Multibase. Otherwise, one cannot access a heterogeneous database in one's familiar data model and data language, unless those model and language happen to be the universal, global data model and language. For example, a relational database user must learn Multibase for the purpose of accessing a hierarchical database in the federation. One may question the wisdom of learning Multibase, in lieu of the hierarchical language, DL/I, in this example, because the relational database user can certainly learn DL/I for greater manipulations of and more direct accesses to the hierarchical database. The benefit of learning Multibase is that Multibase can map its data structure to several equivalent ones, each of which is in a distinct data model. Similarly, Multibase can translate a Multibase transaction into several equivalent transactions, each of which is in a different data language. For instances, a Multibase user may write a Multibase transaction to access a relational database without using SQL and a hierarchical database without using DL/I. Transparent accesses to heterogeneous databases for the Multibase user are achieved in this mapping.

Whenever a new heterogeneous database, i.e., a database in a new data model or data language, is introduced into the federation, it is necessary to extend the Multibase capability by adding a new mapping—one that maps the Multibase data structures to the ones in the new data model or the Multibase transaction into the equivalent

transaction in the new data language or both.

What about existing database users, for example, users of relational, hierarchical, network, functional, and object-oriented databases who desire accesses to databases in other data models? There are two methods: One is to learn the other data models and their corresponding languages; the other is to learn Multibase. Thus, Multibase serves as the global data model and language for the purpose of accessing the heterogeneous databases in other localities. There is no need for the Multibase user to learn local data models and languages in the federation which may be many.

Although the federation of heterogeneous databases by definition supports multiple data models and multiple data languages, the federation in terms of data sharing among heterogeneous databases is monomodel and monolingual. To share data scattered in heterogeneous databases, the user must learn a global data model and use a global data language. An analogy can be found in Esperanto—an artificial language proposed in 1887 for communications and diplomatic exchanges among all people in the world. Obviously, Esperanto is a global language whereas English, Chinese, and other natural languages are local languages. This mapping does not require each database system to support the multimodel and multilingual capabilities. Instead, this mapping requires each system to be bimodel and bilingual, i.e., to support both the global and local data models and languages at each locality, i.e., the local database system. Here, different localities, i.e., different heterogeneous database systems, have different local data models and languages.

On issues of local autonomy, this mapping is similar to the Single-ML-to-Single-ML mapping with perhaps one subtle difference. In the Single-ML-to-Single-ML mapping, access and concurrency controls were specified in a local data language and the enforcement of controls is facilitated by the local access and concurrency control mechanism. However, there was the concern of the global coordination of concurrent accesses to heterogeneous databases, since there was the lack of a global access and concurrency control mechanism for the federation with the Single-ML-to-Single-ML mapping.

Here, in the federation with the Single-ML-to-Multiple-MLs mapping, we have a global data model and a global data language. The question is therefore whether or not the specification for access and concurrency controls of heterogeneous databases can be made in the global data model and language thereby obviating the lack of a global mechanism in carrying out the global specification? The question can be re-phrased as follows: whether or not the global specification of the access and concurrency control requirements can be facilitated by a federation of heterogeneous, local access, and concurrency control mechanisms? The answer is likely no, since the availability of global specifications is no substitute for the absence of a global mechanism. Thus, in

this mapping, issues on deadlocks and indefinite delays over concurrent accesses to heterogeneous databases by "global" transactions persist.

2.3.4 Multiple-Data-Models-and-Languages-to-Multiple-Data-Models-and-Languages Mapping. Finally, we have, for short, *Multiple-MLs-to-Multiple-MLs mapping.* This is perhaps the most complicated approach to data sharing. It may be considered as a two-stage mapping, or two mappings—one after the other. The first one is a Multiple-MLs-to-Single-ML mapping which is followed by a Single-ML-to-Multiple-MLs mapping. Since the single model and language used in the two mappings are intermediate, the net result of the two mappings is therefore a Multiple-MLs-to-Multiple-MLs mapping. Let us first elaborate the two-stage mapping process by way of an example; we then suggest an intermediate data model and data language in the process.

Access transparency is achieved in the same ways that the Multiple-MLs-to-Single-ML mapping and the Single-ML-to-Multiple-MLs mapping have achieved their respective transparent accesses to heterogeneous databases. The question here is how the two mappings are combined in a back-to-back way. This is discussed in the following. The question—why is there the need of an intermediate data model and data language for the mapping?—will be addressed later.

Unlike the Multiple-MLs-to-Single-ML mapping, where the semantics of the single data model and language is more primitive than the semantics of each pair of the multiple data models and languages, the single data model and language used in the first stage of this mapping are rich in semantics. Very likely, all the data structures of databases and language constructs of data languages of federated databases and systems are subsumed by the characterization and specification capabilities of the single data model and language. This is the first difference in using the Multiple-MLs-to-Single-ML mapping here.

The second difference is that there is not a kernel database system here to support the databases in the single data model and to execute transactions written in the single data language. Like the global, universal data model and language discussed in the Single-ML-to-Multiple-MLs mapping, the single data model and language is conceptual and virtual. There is no database system, kernelized or not, to support the model-based databases. In fact, there are no databases in the single data model used in this mapping. The use of the single data model and language is solely for the characterization and specification of the heterogeneous databases in the federation.

The second stage of the process is of the Single-ML-to-Multiple-MLs mapping. Here, the semantic-rich, single data model and language are used as the global, universal data model and language of the mapping. Thus, the second stage of the pro-

cess is identical to the Single-ML-to-Multiple-MLs mapping discussed in Section 2.3.3. A proposal has been made to use the entity-relationship data model and the entity-relationship-based data language (called ER and ERL, respectively), as the single data model and language of this mapping.

In the first stage, for example, if a relational database user desires transparent accesses to a hierarchical database, the user's relational specification of the shared data in a hierarchical database will be transformed by the mapping into an equivalent ER specification, i.e., in the ER data model and language, of the hierarchical database. To access the hierarchical database, the user's relational transactions written, for example, in SQL are translated into equivalent transactions in ERL, i.e., the ER data language. Obviously, in this example, the relational-to-ER schema transformation and the SQL-to-ERL transaction translation have taken place. If the heterogeneity of the federated databases is n, then there are n sets of schema transformers and transaction translators.

In the second and last stage for the same example, the ER schema and ERL transactions created for the relational user are now converted into the equivalent hierarchical schema and DL/I transactions. Again, we need the schema transformer and transaction translator—one for the ER-to-Hierarchical schema transformation and the other for the ERL-to-DL/I transaction translation. Since there are n heterogeneous databases in the federation, there is the need of n sets of schema transformers and transaction translators whose transformations and translations are from the ER and ERL to the local data models and languages, individually. They are the reverse of those n transformers and translators mentioned in the previous paragraph.

In summary, this mapping requires 2n, i.e., (n+n), sets of transformers and translators, as opposed to n(m-1) sets in the Single-ML-to-Single-ML mapping. The additional benefit of this mapping is a significant saving of transformers and translators over the multiplicative benefit of the latter mapping. On the other hand, the number, 2n, as opposed to the number, (n-1), of transaction translators in the Multiple-MLs-to-Single-ML mapping is still two times greater, although it is much less then the number, (n-1)(n-2), of schema transformers needed in the latter mapping. Finally, this mapping, although it utilizes a single data model and language in the two-stage process, it has not reduced the multi-system federation to a kernel-system, i.e., single-system, federation. There are still m heterogeneous database systems in the federation.

The Role of the Intermediate Data Model and Language. Although the use of an intermediate data model and an intermediate data language in this mapping may reduce the number of schema transformers and transaction translators needed for the maximum data sharing in the federation, the original intent for their usage is access and

concurrency controls. Since transparent accesses can only take place after the second stage of the mapping process, the mapping can validate the access request against the integrity constraint, application specificity, and security requirement of the given database at the end of the first stage of the mapping process. To facilitate the validation, the request, constraint, specificity, and requirements are all specified in, i.e., translated into, the intermediate data model and language. As long as the transformation and translation preserve the semantics of the user's request, constraint, specificity, and requirement (specified in the user's data model and language) in the semantics of the intermediate data model and language, the role of the intermediate data model and language in access and concurrency controls of the two-stage process is important and necessary.

We recall in Section 2.3.2, where the Multiple-MLs-to-Single-ML mapping has been elaborated, there is also a common data model and language of federated databases for access and concurrency controls; i.e., the kernel data model and kernel data language, e.g., the attribute-based data model and ABDL. Here, for the Multiple-MLs-to-Multiple-MLs mapping, we use a common data model and language also; i.e., the intermediate data model and language, e.g., the ER data model and ERL. However, there are two important and subtle differences elaborated in the following sections.

Multimodel and Multilingual Capabilities of Federated Database Systems. There can be as many new data models and new data languages as in the other mappings, provided that each time a new set of data model and language is introduced to the federation, two sets of schema transformer and transaction translator are incorporated into the federated systems—one set for the new-ML-to-intermediate-ML transformation and translation and the other for the intermediate-ML-to-new-ML transformation and translation.

However, these multimodel and multiligual capabilities are provided with a more elaborate process, i.e., in two stages whereas in the Single-ML-to-Single-ML mapping process there is only one stage. Even the cross-model accessing capability of the Multiple-MLs-to-Single-ML mapping, although a two-stage process is required in the schema transformation, does not require an additional stage for the transaction translation. This is the first difference of this mapping with others.

Local Autonomy of each Heterogeneous Database in the Federation. The use of a common, intermediate data model and language such as the ER data model and ERL (the object-oriented data model and laguage have also been proposed for this purpose) is desirable for the specification and validation of the access and concurrency controls.

However, unlike the access and concurrency control specification and validation of the Multiple-MLs-to-Single-ML mapping where there is a kernel access and concurrency control mechanism to enforce the specification and to perform the validation, there is not an access and concurrency control mechanism, kernelized or not, present for the ER and ERL specification and validation. To enforce the specification, the ER and ERL must first be translated into the local data model and language of a local database system. To perform the validation, the access and concurrency control mechanism of the local database system must check the translated specification against data in the local database. Thus, the autonomy of each local database is upheld by the local access and control mechanism.

However, for concurrent accesses to databases in different localities and in different database systems, the federation can not rely on a federation of local mechanisms. In other words, it may be necessary to have a common data model and a common data language to specify the integrity constraints, application specificities, and security requirements. It is not sufficient if in the federation there is not a common access and concurrency control mechanism, centralized or distributed, to enforce these specifications. The use of a federation of local mechanisms, in lieu of the common one, will not be sufficient to have deadlock-free accesses to heterogeneous databases in the federation. This is the second difference of between this mapping and the Multiple-MLs-to-Single-ML mapping. In fact, this is the common issue in the multi-system federation. On the other hand, the Multiple-MLs-to-Single-ML mapping does not produce a multi-system federation; it, instead, produces a single-system.

2.4. A Summary of Issues and Approaches to Data Sharing. In Table 2, we tabulate the pros and cons of the five approaches to data sharing of the federated databases. The pros and cons are highlighted against three requirements of data sharing: transparent accesses to heterogeneous databases, multimodel and multilingual capabilities, and the local autonomy of each heterogeneous database in the federation. Where their approaches to one of requirements have about the same effectiveness, we try to show their software complexities in their support of the effectiveness. Obviously, to achieve the same effectiveness, we favor the approach with the least amount of software complexity. In referring to Table 2, we prefer the Multiple-MLs-to-Single-ML mapping for data sharing among federated, heterogeneous databases, because it has more pros and less cons.

3. Concluding Remarks on Data Sharing

In Part I of this tutorial we have pointed out that the proliferation of heterogeneous

Table 2. Merits and Limitations of Five Database Systems

For DATA SHARING	Database Conversion Approach	Single-ML -to- Single-ML Mapping	Multiple-MLs -to- Single-ML Mapping	Single-ML -to- Multiple-MLs Mapping	Multiple-MLs -to- Multiple-MLs Mapping
Copies of same databases	y(demerit)	n	n	n	n
One-time data base reload	y(demerit)	n	y	n	n
Schema trans-formation & management	n	h(demerit)	m	l	l
Transaction transformation	n	h	l(merit)	l	m
T & T process	n	one(merit)	one	one	two
System requirement	multi-system	multi-system	kernel-system	multi-system	multi-system
Kernel or intermediate model/language	n	n	y(merit)	n	y
Learn new model/language	n	n	n	y(demerit)	n
Multimodel & multilingual capabilities	l	l	h(merit)	l	h
Local autonomy	l	h(merit)	h	h	h
Access & concurrency controls	l	l(demerit)	h	l	l

(y = yes; n = no or not applicable; l = low or not available; m = medium; h = high or highly available. In the case of schema transformation and management, the h indicates a high degree of software complexity which is a demerit. On the other hand, h in the case of multimodel and multilingual capabilities is a merit. To aid the reader, we place term "demerit" or "merit" after either a y, h, or l.)

databases and database systems is likely to continue and accelerate. This is partly due to our desire to replace traditional data processing with modern database systems. It is also due to our desire to introduce 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 used for organizational information needs. The interoperability of these heterogeneous databases and database systems becomes necessary for overall data sharing. It also creates the issues of software complexity, access controls, and concurrency controls. In spite of these needs and issues, the organization must uphold the local autonomies of individual database systems and their databases. Otherwise, data sharing will be met with resistances.

In the context of these problems, needs, and complexities, a number of software solutions for data sharing emerges: the Database Conversion approach, Single-ML-to-Single-ML mapping, Multiple-MLs-to-Single-ML mapping, Single-ML-to-Multiple-MLs mapping, and Multiple-MLs-to-Multiple-MLs mapping. These are software-system solutions and are summarized in Table 2. In examining these solutions, the most promising system architecture for data sharing in federated databases and systems appears to be based on the the Multiple-MLs-to-Single-ML mapping. The other solutions have the following issues.

3.1. Technology Issues. To accommodate a high degree of heterogeneity in the federation, there is the necessity of running many heterogeneous database systems and their databases in many separate computers as in the database conversion, Single-ML-to-Single-ML, Single-ML-to-Multiple-MLs, and Multiple-MLs-to-Multiple-MLs approaches. Our technology has not matured enough so that we can configure a large number of computers (of the same or different makes) for data sharing. Consequently, data sharing with any one of the aforementioned software solutions reduces the heterogeneity into a small number, e.g., all the present heterogeneous database systems and their databases in a federation are either bimodel and bilingual.

The belief that a single, all-powerful, all-embracing data model and language can provide transparent accesses to federated, heterogeneous databases and restrict transparent accesses to those who would learn it (as in the Single-ML-to-Multiple-MLs mapping) is not well grounded in technology. Consider the following historical analogy. The development of data structures and their programming languages has had a longer history than the development of data models and their data languages. And yet, we have not seen a single, all-powerful, all-embracing programming language and its data structures which can replace or subsume the multitude of programming languages and their data structures. The requirement of every user to learn a powerful,

new set of data model and language will hinder our effort in data sharing. This kind of transparent accesses to shareable databases is overly restrictive and cumbersome.

Going through an intermediate data model and language for the purpose of accessing heterogeneous databases and systems in the federation (as in Multiple-MLs-to-Multiple-MLs mapping), although technologically feasible, requires a two-stage mapping process and three sets of data models and languages. The software complexity and mapping complications may prevent this mapping to be realized in federated databases and systems.

The development of schema-based database systems, schema transformers, and transaction translators is within the current state of the art. However, it is not clear whether or not the technologist will strive for more schema-based database systems with multimodel and multilingual capabilities. Some technologists are still producing stand-alone, monomodel, monolingual database systems and looking for an all-powerful, all-embracing data model, data language, or database system for all present and foreseeable database applications. To overcome their bias, it is necessary to have a conceptual breakthrough, not just a technology breakthrough.

3.2. Research Issues Although the database conversion approach to data sharing is not the one researchers should pursue, this is the current thrust for developing federated databases and systems. We should assist in alleviating complications arising in this area of research. Thus, some research topics on this approach are also articulated here.

3.2.1. Simultaneous Accesses to Multiple Copies of the Same Database. For the database conversion approach to data sharing where multiple copies of the same database are generated for multiple, separate database systems, there is a need to investigate a multi-system mechanism for access and concurrency controls of simultaneous accesses and manipulations of copies of the same database. The research issue becomes more complicated if these heterogeneous database systems and databases (copies or otherwise) are supported by separate computers. Thus, instead of coming up with a multi-system mechanism for a single computer, we now must come up with a multi-system mechanism on multiple computers.

3.2.2. Schema Transformation, Schema Management, and Transaction Translation. One issue of schema-based systems is schema transformation and the other is transaction translation.

In Multiple-MLs-to-Single-ML mapping, we may be searching for an *"ideal"* kernel data model and language so that the necessary mapping efforts from the existing

and new data models and languages into kernels may be facilitated. Further, the kernel data model and language are conducive to the construction of an efficient kernel database system.

The *transformation algorithms* for a pair of data models and *translation algorithms* for a pair of data languages must be worked out so that data and language semantics are preserved.

For other mappings, we may look for an *automatic transformer-and-translator (t&t) generator.* Since many pairs of schema transformer and transaction translator are needed in the federation, the presence of a "compiler-compiler" will save considerable manual development work for such pairs. What we would like to have is a t&t generator. By giving the formal specification of the data model of a database to the t&t generator, the formal specification of the other data model, and the equivalence table of data constructs (as illustrated in Table 1) equivalent in both data models, the t&t generator can produce a schema transformer which will accept any schema of the database and produce an equivalent schema in the other data model for the database. By giving the formal specification of the data language of a database system to the t&t generator, the formal specification of the data language of the other database system, and the equivalence table of language constructs equivalent in both languages, the t&t generator can produce a transaction translator which will accept any transaction written in the data language of the database system and translate into an equivalent transaction in the data language of the other database system.

The other issue may be *schema and transaction management.* As the number of schemas and transactions increase greatly in the federation, any change in a database may have an effect on the make-up of existing, individual schemas and transactions. In other words, some of the schemas may have to be modified and transactions translated again. The former issue is similar to the issue of view management in the relational database system. We can certainly borrow or generalize some results from there. The latter, can easily be resolved if we make our transaction translators to be *real-time translators* and do not allow any transaction to be cataloged in its object-or-load-module form. In this way, every transaction will have to be translated in real-time, no matter how frequent or long they will be used by the database system. Our research is therefore to seek the most efficient and effective real-time translation techniques. Together, these issues are also referred to as the *integration* issues of schemas and transactions.

In Multiple-MLs-to-Single-ML mapping, there is an *access and concurrency control mechanism in the kernel database system.* Since all the heterogeneous databases in the federation are supported by the kernel database system, the research issue is therefore whether the kernel mechanism can uphold the individual autonomy of each heterogeneous database of the federation.

In the other three mappings, the absence of a kernelized access and concurrency control mechanism is evident. Further, they have resulted a multi-system architecture. To avoid deadlocks or indefinite delays in concurrent accesses to heterogeneous databases, we need some kind of coordination and scheduling among the autonomous local controls. Can we forgo the construction of a kernelized, multi-system mechanism for the coordination and scheduling , and rely solely on some kind of global specifications for local controls? If the answer is no, then what kind of "global" mechanism or mechanisms should we propose? Or, is there no hope to coordinate autonomous local controls in a multi-system environment by an overall mechanism? These are the open questions.

Postscripts

The author would like to thank (Heimbigner, 1985) for the articulation of the notion of *federated* architecture. Although the author is told that Manning and his colleagues have used the term for distributed operating systems in a paper in 70's, the author is not aware of that article. In any case, on the basis of (Hsiao, 1989), a precise definition and necessary requirements for *federated databases and systems* are given in this tutorial for the first time. An extended abstract of the tutorial is published in (Hsiao, 1990).

In order to review the entire field of federated databases and systems and cite most of the past work on the subject matter, a taxonomy of issues and solutions is needed, so that we may place our problems and results in perspective. The taxonomy was first proposed in (Hsiao, 1989) and expanded herein. We make no citation of papers on classical data models and languages such as relational data model and SQL. Nor do we make references to papers on commercial products, classical work, and development efforts. We only refer to published research papers, technical reports, and theses. Since the database conversion approach to data sharing is a part of the present technology, we have no reference on it either. Thus, papers in the proceedings, books, and journals cited below are for the other four mappings.

For the Single-ML-to-Multiple-MLs mapping, see (Rosenberg, 1982). For Multiple-MLs-to-Multiple-MLs mapping, read (Cardenas, 1987). For Multiple-MLs-to-Single-ML mapping, see (Demurjian, 1985, 1987, 1988). Since the Single-ML-to-Single-ML mapping is a special case of other mappings, there is no need to have its own references. What we need are references on the many, specific schema transformations and transaction translations. On the relational data model and/or language to/from the hierarchical data model and/or language, see (Meng, 1990). On the relational data model/language to/from the network data model/language, see (Demo,

1985; Katz, 1982; Larson, 1983). On the functional data model/language to/from the attribute-based data model/language, see (Buneman, 1979, 1984; Goisman, 1985; Mack, 1992; Shipman, 1981). On the objected-oriented data model/language to/from relational data model/language, see (King, 1984; Wu, 1989: Zaniolo, 1984). On the objected-oriented data model/language to attribute-based data model/language, see (Hogan, 1989). On the relational data model/language to attribute-based data model/language, see (Kloepping, 1985; Rollins, 1984). On the hierarchical data model/language to attribute-based data model and language, see (Benson, 1985; Weishar, 1984). On the network data model/language to attribute-based data model/language, see (Antony, 1985; Wortherly, 1985). On cross-model accessing capability, for instance, accessing hierarchical databases with SQL transactions, see (Zawis, 1987).

On access and concurrency control mechanisms for federated databases and systems, see (Du, 1989a, 1989b; Elmagarmid, 1990). There are also several papers on the same issue in (Gupta, 1989; Scheuermann, 1989). On the use of a kernel access control mechanism for supporting external security requirements and policies such as the relational views and the U.S. DoD multilevel security, the reader may refer to (Hoppenstand, 1989; Hsiao, 1991).

There are many related issues on federated databases and systems, the reader is encouraged to browse through some of the articles in (Gupta, 1989; Scheuermann, 1989), and some of the recent publications such as (Kamel, 1990; Wang, 1990).

It is important to note that the conclusion and references in this paper relate to Part I of this tutorial on data sharing in federated databases and systems. Part II concerns resource consolidation in federated databases and systems which has its own conclusion and references. Part II will appear in a future issue of this journal.

Acknowledgments

The work reported here is supported by funds from NPS, NPMTC, and NRL. The author would like to thank the referees for their recommended changes.

References

Antony, J.A. and Billings, A.J. Implementation of the Network/CODASY-DML Interface for the Multilingual Database System, Master's Thesis in Computer Science, Naval Postgraduate School, Monterey, CA, December 1985.

Benson, T.P. and Wentz, G.L. Implementation of a Hierarchical/DL/I Interface for the Multilingual Database System, Master's Thesis in Computer Science, Naval Postgraduate School, Monterey, CA, June 1985.

Buneman, O.P. and Frankel, R.E. FQL—A Functional Query Language, *Proceedings of the 1979 ACM SIGMOD Conference,* New York: ACM Press, 1979.

Buneman, O.P. and Nikhil, R. The Functional Data Model and its Uses for Interaction with Databases, *Conceptual Modeling,* New York: Springer-Verlag, 1984.

Cardenas, A.F. Heterogeneous Distributed Database Management: The HDDBMS, *Proceedings of the IEEE,* 75:5, 1987.

Demo, G.B. and Kundu, S. Analysis of the Context Dependency of CODASYL Find-Statements with Application to Database Program Conversion, *Proceedings of the 1985 ACM SIGMOD Conference,* New York: ACM Press, 1985.

Demurjian, S.A. and Hsiao, D.K. New Directions in Database-Systems Research and Development, *Proceedings of the International Symposium on New Directions in Computing,* Washington, D.C.: IEEE Computer Society Press, 1985.

Demurjian, S.A. and Hsiao, D.K. The Multilingual Database System, *Proceedings of the Third IEEE International Conference on Data Engineering,* Los Angeles, CA, 1987.

Demurjian, S.A. and Hsiao, D.K. Towards a Better Understanding of Data Models through the Multilingual Database Systems, *IEEE Transactions on Software Engineering,* 14:7, 1988.

Du, W. and Elmagarmid, A.K. Quasi Serializability: A Correctness Criterion for Global Concurrency Control in InterBase, *Proceedings of the Fifteenth International Conference on Very Large Databases,* Brisbane, Australia, 1989*a*.

Du, W. et al. Effects of Local Autonomy on Global Concurrency Control in Heterogeneous Distributed Database Systems, *Proceedings of the Second International Conference on Data and Knowledge Systems for Manufacturing and Engineering,* New York, 1989*b*.

Elmagarmid, A.K. and Du, W. A Paradigm for Concurrency Control in Heterogeneous Distributed Database Systems, *Proceedings of the Sixth IEEE International Conference of Data Engineering,* Washington, D.C., 1990.

Goisman, P.L. Design and Analysis of a Complete Functional/Daplex Interface for the Multilingual Database System, Master's Thesis in Computer Science, Naval Postgraduate School, Monterey, CA, December 1985.

Gupta, A., ed. Integration of Information Systems: Bridging Heterogeneous Databases, The IEEE Press Selected Reprint Series, Washington, D.C.: IEEE Press, 1989.

Heimbigner, D. and McLeod, D. A Federated Architecture for Information Management, *ACM Transactions on Office Information Systems,* 3:3, 1985.

Hogan, T.R. Interconnection of the Graphics Language for Database System (GLAD) to the Multilingual, Multimodel, and Multibackend Database System over an Ethernet Network, Master's Thesis in Computer Science, Naval Postgraguate School, December 1989.

Hoppenstand, G.S. and Hsiao, D.K. Secure Access Control with High Access Precision: An Efficient Approach to Multilevel Security. In: Landwehr, C.E., ed. *Database Security, II—Status and Prospects,* New York: North-Holland, 1989.

Hsiao, D.K. and Kamel, M.N. Heterogeneous Databases: Proliferations, Issues, and Solutions, *IEEE Transactions on Knowledge and Data Engineering,* 1:1, 1989.

Hsiao, D.K., Kamel, M.N., and Wu, C.T. The Federated Databases and System–A New Generation of Advanced Database Systems, *Proceedings of International Conference on Database and Systems Applications (DAXA 90),* Vienna, Austria, 1990.

Hsiao, D.K., Kohler, M.J., and Stround, S.W. Query Modification as a Means of Controlling Accesses to Multilevel Secure Databases. In: Landwehr, C.E., ed. *Database Security, VI—Status and Prospects,* New York: North-Holland, 1991.

Kamel, M.N. and Hsiao, D.K. Interoperability and Integration Issues in Heterogeneous Database Environment, *Proceedings of 1990 Navy Database Symposium (Database '90),* San Diego, CA, 1990.

Katz, R.H. and Wong, E. Decompiling CODASYL DML into Relational Queries, *ACM Transactions on Database Systems,* 7:1, 1982.

King, R. A Database Management System Based on an Object-Oriented Model, *Proceedings of the First Workshop on Expert Database Systems,* Charleston, SC, 1984.

Kloepping, G.R. and Mack, J.F. Implementation of a Relational/SQL Interface for the Multilingual Database System, Master's Thesis in Computer Science, Naval Postgraguate School, June 1985.

Larson, J.A. Bridging the Gap between Network and Relational Database Management Systems, *Computer,* 16:9, 1983.

Mack, S.B. Implementation of a Functional/Daplex Interface for the Multilingual Database System, Master's Thesis in Computer Science, Naval Postgraduate School, March 1992.

Meng, W., et al. Transformation of Relational Queries to Hierarchical Queries, Technical Report, Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, 1990.

Rollins, R. Design and Analysis of a Complete Relational/SQL Interface for the Multilingual Database System. Master's Thesis in Computer Science, Naval Postgraduate School, Monterey, CA, June 1984.

Rosenberg, R.L. and Landers, T. An Overview of MULTIBASE. In: Schneider, H.J., ed. *Distributed Databases,* New York: North-Holland, 1982.

Scheuermann, P., Yu, C., Elmagarmid, A., Garcia-Molina, H., Manola, F., McLeod, D., Rosenthal, A., and Templeton, M., eds. Position Papers of the 1989 NSF-Sponsored Workshop on Heterogeneous Databases, The Northwestern University, 1989.

Shipman, D.W. The Functional Data Model and the Data Language Daplex, *ACM Transactions on Database Systems,* 6:1, 1981.

Wang, Y.R. and Madnick, S.E. A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective, *Proceedings of the Sixteenth International Conference on Very Large Data Bases,* Barcelona, Spain, 1990.

Weishar, D.J. Design and Analysis of a Complete Hierarchical Interface for the Multilingual Database System, Master's Thesis in Computer Science, Naval Postgraduate School, Monterey, CA, 1984.

Wortherly, C.R. Design and Analysis of a Network/CODASYL-DML Interface for the Multilingual Database System, Master's Thesis in Computer Science, Naval Postgraguate School, Monterey, CA, December 1985.

Wu, C.T. and D. K. Hsiao, Implementation of an Object-Oriented Data Language: GLAD, *Proceedings of the IFIP WG 2.6 Visual Database Conference,* Tokyo, Japan, 1989.

Zaniolo, C., et al. Object-Oriented Database Systems and Knowledge Systems, *Proceedings of the First Workshop on Expert Database Systems,* Charleston, SC, 1984.

Zawis, J. Design and Implementation of a Cross-Model Accessing Capability: Accessing Hierarchical Databases Relationally—for the Multimodel and Multilingual Database System. Master's Thesis in Computer Science, Naval Postgraduate School, Monterey, CA, December 1987.