Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases[1]

AMIT P. SHETH

Bellcore, 1J-210, 444 Hoes Lane, Piscataway, New Jersey 08854

JAMES A. LARSON

Intel Corp., HF3-02, 5200 NE Elam Young Pkwy., Hillsboro, Oregon 97124

A federated database system (FDBS) is a collection of cooperating database systems that are autonomous and possibly heterogeneous. In this paper, we define a reference architecture for distributed database management systems from system and schema viewpoints and show how various FDBS architectures can be developed. We then define a methodology for developing one of the popular architectures of an FDBS. Finally, we discuss critical issues related to developing and operating an FDBS.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*methodologies*; D.2.10 [**Software Engineering**]: Design; H.0 [**Information Systems**]: General; H.2.0 [**Database Management**]: General; H.2.1 [**Database Management**]: Logical Design—*data models, schema and subschema*; H.2.4 [**Database Management**]: Systems; H.2.5 [**Database Management**]: Heterogeneous Databases; H.2.7 [**Database Management**]: Database Administration

General Terms: Design, Management

Additional Key Words and Phrases: Access control, database administrator, database design and integration, distributed DBMS, federated database system, heterogeneous DBMS, multidatabase language, negotiation, operation transformation, query processing and optimization, reference architecture, schema integration, schema translation, system evolution methodology, system/schema/processor architecture, transaction management

INTRODUCTION

Federated Database System

A *database system* (DBS) consists of software, called a database management system (DBMS), and one or more databases that it manages. A *federated database system* (FDBS) is a collection of cooperating but autonomous component database systems (DBSs). The component DBSs are

[1] The views and conclusions in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Bellcore, Intel Corp., or the authors' past or present affiliations. It is the policy of Bellcore to avoid any statements of comparative analysis or evaluation of vendors' products. Any mention of products or vendors in this document is done where necessary for the sake of scientific accuracy and precision, or for background information to a point of technology analysis, or to provide an example of a technology for illustrative purposes and should not be construed as either positive or negative commentary on that product or that vendor. Neither the inclusion of a product or a vendor in this paper nor the omission of a product or a vendor should be interpreted as indicating a position or opinion of that product or vendor on the part of the author(s) or of Bellcore.

CONTENTS

integrated to various degrees. The software that provides controlled and coordinated manipulation of the component DBSs is called a federated database management system (FDBMS) (see Figure 1).

Both databases and DBMSs play important roles in defining the architecture of an FDBS. Component database refers to a database of a component DBS. A component DBS can participate in more than one federation. The DBMS of a component DBS, or component DBMS, can be a centralized or distributed DBMS or another FDBMS. The component DBMSs can differ in such aspects as data models, query languages, and transaction management capabilities.

One of the significant aspects of an FDBS is that a component DBS can continue its local operations and at the same time participate in a federation. The integration of component DBSs may be managed either by the users of the federation or by the administrator of the FDBS together with the administrators of the component DBSs. The amount of integration depends on the needs of federation users and desires of the administrators of the component DBSs to participate in the federation and share their databases. The term *federated database system* was coined by Hammer and McLeod [1979] and Heimbigner and McLeod [1985]. Since its introduction, the term has been used for several different but related DBS architectures. As explained in this Introduction, we use the term in its broader context and include additional architectural alternatives as examples of the federated architecture.

The concept of federation exists in many contexts. Consider two examples from the political domain—the United Nations (UN) and the Soviet Union. Both entities exhibit varying levels of autonomy and heterogeneity among the components (sovereign nations and the republics, respectively). The autonomy and heterogeneity is greater in the UN than in the Soviet Union. The power of the federation body (the General Assembly of the UN and the central government of the Soviet Union, respectively) with respect to its components in the two cases is also different. Just as people do not agree on an ideal model or the utility of a federation for the political bodies and the governments, the database context has no single or ideal model of federation. A key characteristic of a federation, however, is the cooperation among independent systems. In terms of an FDBS, it is reflected by controlled and sometimes limited integration of autonomous DBSs.

The goal of this survey is to discuss the application of the federation concept for managing existing heterogeneous and au-

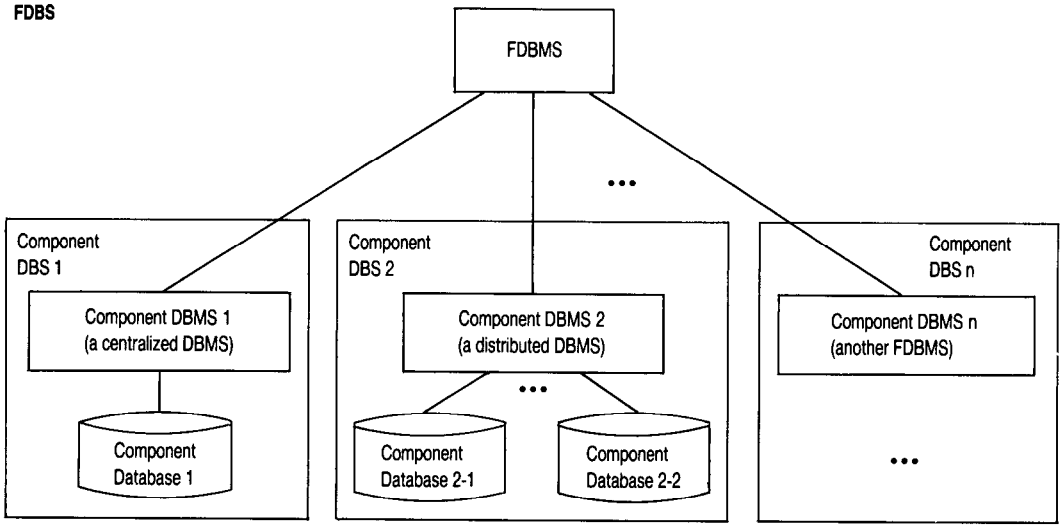

Figure 1. An FDBS and its components.

tonomous DBSs. We describe various architectural alternatives and components of a federated database system and explore the issues related to developing and operating such a system. The survey assumes an understanding of the concepts in basic database management textbooks [Ceri and Pelagatti 1984; Date 1986; Elmasri and Navathe 1989; Tsichritzis and Lochovsky 1982] such as data models, the ANSI/SPARC schema architecture, database design, query processing and optimization, transaction management, and distributed database management.

Characteristics of Database Systems

Systems consisting of multiple DBSs, of which FDBSs are a specific type, may be characterized along three orthogonal dimensions: distribution, heterogeneity, and autonomy. These dimensions are discussed below with an intent to classify and define such systems. Another characterization based on the dimensions of the networking environment [single DBS, many DBSs in a local area network (LAN), many DBSs in a wide area network (WAN), many networks], update related functions of participating DBSs (e.g., no update, nonatomic updates, atomic updates), and the types of heterogeneity (e.g., data models, transaction management strategies) has been proposed by Elmagarmid [1987]. Such a characterization is particularly relevant to the study and development of transaction management in FDBMS, an aspect of FDBS that is beyond the scope of this paper.

Distribution

Data may be distributed among multiple databases. These databases may be stored on a single computer system or on multiple computer systems, co-located or geographically distributed but interconnected by a communication system. Data may be distributed among multiple databases in different ways. These include, in relational terms, vertical and horizontal database partitions. Multiple copies of some or all of the data may be maintained. These copies need not be identically structured.

Benefits of data distribution, such as increased availability and reliability as well as improved access times, are well known [Ceri and Pelagatti 1984]. In a distributed DBMS, distribution of data may be induced; that is, the data may be deliberately distributed to take advantage of these benefits. In the case of FDBS, much of the data distribution is due to the existence of multiple DBSs before an FDBS is built.

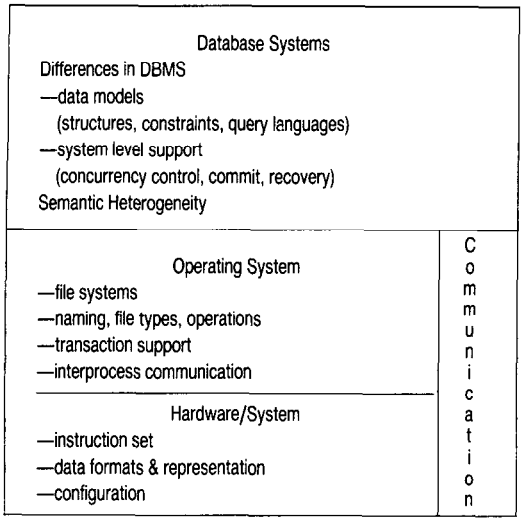

Figure 2. Types of heterogeneities.

Heterogeneity

Many types of heterogeneity are due to technological differences, for example, differences in hardware, system software (such as operating systems), and communication systems. Researchers and developers have been working on resolving such heterogeneities for many years. Several commercial distributed DBMSs are available that run in heterogeneous hardware and system software environments.

The types of heterogeneities in the database systems can be divided into those due to the differences in DBMSs and those due to the differences in the semantics of data (see Figure 2).

Heterogeneities due to Differences in DBMSs

An enterprise may have multiple DBMSs. Different organizations within the enterprise may have different requirements and may select different DBMSs. DBMSs purchased over a period of time may be different due to changes in technology. Heterogeneities due to differences in DBMSs result from differences in data models and differences at the system level. These are described below. Each DBMS has an underlying data model used to define data structures and constraints. Both representation (structure and constraints) and language aspects can lead to heterogeneity.

- **Differences in structure:** Different data models provide different structural primitives [e.g., the information modeled using a relation (table) in the relational model may be modeled as a record type in the CODASYL model]. If the two representations have the same information content, it is easier to deal with the differences in the structures. For example, address can be represented as an entity in one schema and as a composite attribute in another schema. If the information content is not the same, it may be very difficult to deal with the difference. As another example, some data models (notably semantic and object-oriented models) support generalization (and property inheritance) whereas others do not.
- **Differences in constraints:** Two data models may support different constraints. For example, the set type in a CODASYL schema may be partially modeled as a referential integrity constraint in a relational schema. CODASYL, however, supports insertion and retention constraints that are not captured by the referential integrity constraint alone. Triggers (or some other mechanism) must be used in relational systems to capture such semantics.
- **Differences in query languages:** Different languages are used to manipulate data represented in different data models. Even when two DBMSs support the same data model, differences in their query languages (e.g., QUEL and SQL) or different versions of SQL supported by two relational DBMSs could contribute to heterogeneity.

Differences in the system aspects of the DBMSs also lead to heterogeneity. Examples of system level heterogeneity include differences in transaction management primitives and techniques (including concurrency control, commit protocols, and recovery), hardware and system

software requirements, and communication capabilities.

Semantic Heterogeneity

Semantic heterogeneity occurs when there is a disagreement about the meaning, interpretation, or intended use of the same or related data. A recent panel on semantic heterogeneity [Cercone et al. 1990] showed that this problem is poorly understood and that there is not even an agreement regarding a clear definition of the problem. Two examples to illustrate the semantic heterogeneity problem follow.

Consider an attribute MEAL_COST of relation RESTAURANT in database DB1 that describes the average cost of a meal per person in a restaurant *without* service charge and tax. Consider an attribute by the same name (MEAL_COST) of relation BOARDING in database DB2 that describes the average cost of a meal per person *including* service charge and tax. Let both attributes have the same syntactic properties. Attempting to compare attributes DB1.RESTAURANTS.MEAL_COST and DB2.BOARDING.MEAL_COST is misleading because they are semantically heterogeneous. Here the heterogeneity is due to differences in the definition (i.e., in the meaning) of related attributes [Litwin and Abdellatif 1986].

As a second example, consider an attribute GRADE of relation COURSE in database DB1. Let COURSE.GRADE describe the grade of a student from the set of values {A, B, C, D, F}. Consider another attribute SCORE of relation CLASS in database DB2. Let SCORE denote a normalized score on the scale of 0 to 10 derived by first dividing the weighted score of all exams on the scale of 0 to 100 in the course and then rounding the result to the nearest half-point. DB1.COURSE.GRADE and DB2.CLASS.SCORE are semantically heterogeneous. Here the heterogeneity is due to different precision of the data values taken by the related attributes. For example, if grade C in DB1.COURSE.GRADE corresponds to a weighted score of all exams between 61 and 75, it may not be possible to correlate it to a score in DB2.CLASS.SCORE because both 73 and 77 would have been represented by a score of 7.5.

Detecting semantic heterogeneity is a difficult problem. Typically, DBMS schemas do not provide enough semantics to interpret data consistently. Heterogeneity due to differences in data models also contributes to the difficulty in identification and resolution of semantic heterogeneity. It is also difficult to decouple the heterogeneity due to differences in DBMSs from those resulting from semantic heterogeneity.

Autonomy

The organizational entities that manage different DBSs are often autonomous. In other words, DBSs are often under separate and independent control. Those who control a database are often willing to let others share the data only if they retain control. Thus, it is important to understand the aspects of *component autonomy* and how they can be addressed when a component DBS participates in an FDBS.

A component DBS participating in an FDBS may exhibit several types of autonomy. A classification discussed by Veijalainen and Popescu-Zeletin [1988] includes three types of autonomy: design, communication, and execution. These and an additional type of component autonomy called association autonomy are discussed below.

Design autonomy refers to the ability of a component DBS to choose its own design with respect to any matter, including

(a) The data being managed (i.e., the Universe of Discourse),
(b) The representation (data model, query language) and the naming of the data elements,
(c) The conceptualization or semantic interpretation of the data (which greatly contributes to the problem of semantic heterogeneity),

(d) Constraints (e.g., semantic integrity constraints and the serializability criteria) used to manage the data,

(e) The functionality of the system (i.e., the operations supported by system),

(f) The association and sharing with other systems (see association autonomy below), and

(g) The implementation (e.g., record and file structures, concurrency control algorithms).

Heterogeneity in an FDBS is primarily caused by design autonomy among component DBSs.

The next two types of autonomy involve the DBMS of a component DBS. *Communication autonomy* refers to the ability of a component DBMS to decide whether to communicate with other component DBMSs. A component DBMS with communication autonomy is able to decide when and how it responds to a request from another component DBMS.

Execution autonomy refers to the ability of a component DBMS to execute local operations (commands or transactions submitted directly by a local user of the component DBMS) without interference from external operations (operations submitted by other component DBMSs or FDBMSs) and to decide the order in which to execute external operations. Thus, an external system (e.g., FDBMS) cannot enforce an order of execution of the commands on a component DBMS with execution autonomy. Execution autonomy implies that a component DBMS can abort any operation that does not meet its local constraints and that its local operations are logically unaffected by its participation in an FDBS. Furthermore, the component DBMS does not need to inform an external system of the order in which external operations are executed and the order of an external operation with respect to local operations. Operationally, a component DBMS exercises its execution autonomy by treating external operations in the same way as local operations.

Association autonomy implies that a component DBS has the ability to decide whether and how much to share its functionality (i.e., the operations it supports) and resources (i.e., the data it manages) with others. This includes the ability to associate or disassociate itself from the federation and the ability of a component DBS to participate in one or more federations. Association autonomy may be treated as a part of the design autonomy or as an autonomy in its own right. Alonso and Barbara [1989] discuss the issues that are relevant to this type of autonomy.

A subset of the above types of autonomy were also identified by Heimbigner and McLeod [1985]. Du et al. [1990] use the term *local autonomy* for the autonomy of a component DBS. They define two types of local autonomy requirements: operation autonomy requirements and service autonomy requirements. *Operation autonomy requirements* relate to the ability of a component DBS to exercise control over its database. These include the requirements related to design and execution autonomy. *Service autonomy requirements* relate to the right of each component DBS to make decisions regarding the services it provides to other component DBSs. These include the requirements related to association and communication autonomy. Garcia-Molina and Kogan [1988] provide a different classification of the types of autonomy. Their classification is particularly relevant to the operating system and transaction management issues.

The need to maintain the autonomy of component DBSs and the need to share data often present conflicting requirements. In many practical environments, it may not be desirable to support the autonomy of component DBSs fully. Two examples of relaxing the component autonomy follow:

- Association autonomy requires that each component DBS be free to associate or disassociate itself from the federation. This would require that the FDBS be designed so that its existence and operation are not dependent on any single component DBS. Although this may be a desirable design goal, the FDBS may moderate it by requiring that the entry or departure of a component DBS must be based on an agreement between the

federation (i.e., its representative entity such as the administrator of the FDBS) and the component DBS (i.e., the administrator of a component DBS) and cannot be a unilateral decision of the component DBS.

- Execution autonomy allows a component DBS to decide the order in which external and local operations are performed. Futhermore, the component DBS need not inform the external system (e.g., FDBS) of this order. This latter aspect of autonomy may, however, be relaxed by informing the FDBS of the order of transaction execution (or transaction wait-for graph) to allow simpler and more efficient management of global transactions.

Taxonomy of Multi-DBMS and Federated Database Systems

A DBS may be either centralized or distributed. A *centralized DBS* system consists of a single centralized DBMS managing a single database on the same computer system. A *distributed DBS* consists of a single distributed DBMS managing multiple databases. The databases may reside on a single computer system or on multiple computer systems that may differ in hardware, system software, and communication support. A *multidatabase system* (MDBS) supports operations on multiple component DBSs. Each component DBS is managed by (perhaps a different) component DBMS. A component DBS in an MDBS may be centralized or distributed and may reside on the same computer or on multiple computers connected by a communication subsystem. An MDBS is called a *homogeneous MDBS* if the DBMSs of all component DBSs are the same; otherwise it is called a *heterogeneous MDBS*. A system that only allows periodic, nontransaction-based exchange of data among multiple DBMSs (e.g., EXTRACT [Hammer and Timmerman 1989]) or one that only provides access to multiple DBMSs *one at a time* (e.g., no joins across two databases) is not called an MDBS. The former is a *data exchange system*; the latter is a *remote DBMS interface* [Sheth 1987a].

Different architectures and types of FDBSs are created by different levels of integration of the component DBSs and by different levels of global (federation) services. We will use the taxonomy shown in Figure 3 to compare the architectures of various research and development efforts. This taxonomy focuses on the autonomy dimension. Other taxonomies are possible by focusing on the distribution and heterogeneity dimensions. Some recent publications discussing various architectures or different taxonomies include Eliassen and Veijalainen [1988], Litwin and Zeroual [1988], Ozsu and Valduriez [1990], and Ram and Chastain [1989].

MDBSs can be classified into two types based on the autonomy of the component DBSs: nonfederated database systems and federated database systems. A *nonfederated database system* is an integration of component DBMSs that are *not* autonomous. It has only one level of management,[2] and all operations are performed uniformly. In contrast to a federated database system, a nonfederated database system does not distinguish local and nonlocal users. A particular type of nonfederated database system in which all databases are fully integrated to provide a single global (sometimes called *enterprise* or *corporate*) schema can be called a *unified MDBS*. It logically appears to its users like a distributed DBS.

A *federated database system* consists of component DBSs that are autonomous yet participate in a federation to allow partial and controlled sharing of their data. Association autonomy implies that the component DBSs have control over the data they manage. They cooperate to allow different degrees of integration. There is no centralized control in a federated architecture because the component DBSs (and their database administrators) control access to their data.

FDBS represents a compromise between no integration (in which users must explicitly interface with multiple autonomous databases) and total integration (in which

[2] This definition may be diluted to include two levels of management, where the global level has the authority for controlling data sharing.

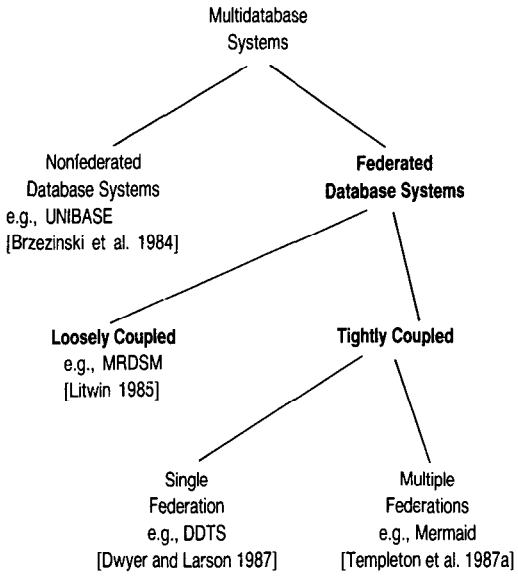

Figure 3. Taxonomy of multidatabase systems.

autonomy of each component DBS is sacrificed so that users can access data through a single global interface but cannot directly access a DBMS as a local user). The federated architecture is well suited for migrating a set of autonomous and standalone DBSs (i.e., DBSs that are not sharing data) to a system that allows partial and controlled sharing of data without affecting existing applications (and hence preserving significant investment in existing application software).

To allow controlled sharing while preserving the autonomy of component DBSs and continued execution of existing applications, an FDBS supports two types of operations: local and global (or federation). This dichotomy of local and global operations is an essential feature of an FDBS. Global operations involve data access using the FDBMS and may involve data managed by multiple component DBSs. Component DBSs must grant permission to access the data they manage. Local operations are submitted to a component DBS directly. They involve only data in that component DBS. A component DBS, however, does not need to distinguish between local and global operations. In most environments, the FDBS will also be heterogeneous, that is, will consist of heterogeneous component DBSs. In the rest of this paper, we will use the term FDBS to describe a heterogeneous distributed DBS with autonomy of component DBSs.

FDBSs can be categorized as loosely coupled or tightly coupled based on who manages the federation and how the components are integrated. An FDBS is *loosely coupled* if it is the *user's* responsibility to create and maintain the federation and there is no control enforced by the federated system and its administrators. Other terms used for loosely coupled FDBSs are *interoperable database system* [Litwin and Abdellatif 1986] and *multidatabase system* [Litwin et al. 1982].[3] A federation is *tightly coupled* if the federation and its administrator(s) have the responsibility for creating and maintaining the federation and actively control the access to component DBSs. Association autonomy dictates that, in both cases, sharing of any part of a component database or invoking a capability (i.e., an operation) of a component DBS is controlled by the administrator of the component DBS.

A federation is built by a selective and controlled integration of its components. The activity of developing an FDBS results in creating a federated schema upon which operations (i.e., query and/or updates) are performed. A loosely coupled FDBS always supports multiple federated schemas. A tightly coupled FDBS may have one or more federated schemas. A tightly coupled FDBS is said to have *single federation* if it allows the creation and management of only one federated schema.[4] Having a single

[3] The term multidatabase has been used by different people to mean different things. For example, Litwin [1985] and Rusinkiewicz et al. [1989] use the term multidatabase to mean loosely coupled FDBS (or interoperable system) in our taxonomy; Ellinghaus et al. [1988] and Veijalainen and Popescu-Zeletin [1988] use it to mean client-server type of FDBS in our taxonomy; and Dayal and Hwang [1984], Belcastro et al. [1988], and Breitbart and Silberschatz [1988] use it to mean tightly coupled FDBS in our taxonomy.

[4] Note that a tightly coupled FDBS with a single federated schema is not the same as a unified MDBS but is a special case of the latter. It espouses the federation concepts such as autonomy of component DBMSs, dichotomy of operations, and controlled sharing that a unified MDBS does not.

federated schema helps in maintaining uniformity in semantic interpretation of the integrated data. A tightly coupled FDBS is said to have *multiple federations* if it allows the creation and management of multiple federated schemas. Having multiple federated schemas readily allows multiple integrations of component DBSs. Constraints involving multiple component DBS, however, may be difficult to enforce. An organization wanting to exercise tight control over the data (treated as a corporate resource) and the enforcement of constraints (including the so-called business rules) may choose to allow only one federated schema.

The terms *federated database system* and *federated database architecture* were introduced by Heimbigner and McLeod [1985] to mean "collection of components to unite loosely coupled federation in order to share and exchange information" and "an organization model based on equal, autonomous databases, with sharing controlled by explicit interfaces." The multidatabase architecture of Litwin et al. [1982] shares many features of the above architecture. These definitions include what we have defined as loosely coupled FDBSs. The key FDBS concepts, however, are autonomy of components, and partial and controlled sharing of data. These can also be supported when the components are tightly coupled. Hence we include both loosely and tightly coupled FDBSs in our definition of FDBSs.

MRDSM [Litwin 1985], OMNIBASE [Rusinkiewicz et al. 1989], and CALIDA [Jacobson et al. 1988] are examples of loosely coupled FDBSs. In CALIDA, federated schemas are generated by a database administrator rather than users as in other loosely coupled FDBSs. Users must be relatively sophisticated in other loosely coupled FDBSs to be able to define schemas/views over multiple component DBSs. SIRIUS-DELTA [Litwin et al. 1982] and DDTS [Dwyer and Larson 1987] can be categorized as tightly coupled FDBSs with single federation. Mermaid® [Templeton et al. 1987b] and Multibase [Landers and Rosenberg 1982] are examples of tightly coupled FDBSs with multiple federations.

® Mermaid is a trademark of Unisys Corporation.

A type of FDBS architecture called the *client-server architecture* has been discussed by Ge et al. [1987] and Eliassen and Veijalainen [1988]. In such a system, there is an explicit contract between a client and one or more servers for exchanging information through predefined transactions. A client-server system typically does not allow ad hoc transactions because the server is designed to respond to a set of predefined requests. The schema architecture of a client-server system is usually quite simple. The schema of each server is directly mapped to the schema of the client. Thus the client-server architecture can be considered to be a tightly coupled one for FDBS with multiple federations.

Scope and Organization of this Paper

Issues involved in managing an FDBS deal with distribution, heterogeneity, and autonomy. Issues related to distribution have been addressed in past research and development efforts on distributed DBMSs. We will concentrate on the issues of autonomy and heterogeneity. Recent surveys on the related topics include Barker and Ozsu [1988]; Litwin and Zeroual [1988]; Ram and Chastain [1989], and Siegel [1987].

The remainder of this paper is organized as follows. In Section 1 we discuss a reference architecture for DBSs. Two types of system components—processors and schemas—are particularly applicable to FDBSs. In Section 2 we use the processors and schemas to define various FDBS architectures. In Section 3 we discuss the phases in an FDBS evolution process. We also discuss a methodology for developing a tightly coupled FDBS with multiple federations. In Section 4 we discuss four important tasks in developing an FDBS: schema translation, access control, negotiation, and schema integration. In Section 5 we discuss four tasks relevant to operating an FDBS: query formulation, command transformation, query processing and optimization, and transaction management. Section 6 summarizes and discusses issues that need further research and development. The paper ends with references, a comprehensive bibliography, a glossary of the terms

used throughout this paper, and an appendix comparing some features of relevant prototype efforts.

1. REFERENCE ARCHITECTURE

A reference architecture is necessary to clarify the various issues and choices within a DBS. Each component of the reference architecture deals with one of the important issues of a database system, federated or otherwise, and allows us to ignore details irrelevant to that issue. We can concentrate on a small number of issues at a time by analyzing a single component. A reference architecture provides the framework in which to understand, categorize, and compare different architectural options for developing federated database systems. Section 1.1 discusses the basic system components of a reference architecture. Section 1.2 discusses various types of processors and the operations they perform on commands and data. Section 1.3 discusses a schema architecture of a reference architecture. Other reference architectures described in the literature include Blakey [1987], Gligor and Luckenbaugh [1984], and Larson [1989].

1.1 System Components of a Reference Architecture

A reference architecture consists of various system components. Basic types of system components in our reference architecture are as follows:

- **Data:** Data are the basic facts and information managed by a DBS.
- **Database:** A database is a repository of data structured according to a data model.
- **Commands:** Commands are requests for specific actions that are either entered by a user or generated by a processor.
- **Processors:** Processors are software modules that manipulate commands and data.
- **Schemas:** Schemas are descriptions of data managed by one or more DBMSs. A schema consists of schema objects and their interrelationships. Schema objects are typically class definitions (or data structure descriptions) (e.g., table definitions in a relational model), and entity types and relationship types in the entity-relationship model.
- **Mappings:** Mappings are functions that correlate the schema objects in one schema to the schema objects in another schema.

These basic components can be combined in different ways to produce different data management architectures. Figure 4 illustrates the iconic symbols used for each of these basic components. The reasons for choosing these components are as follows:

- Most centralized, distributed, and federated database systems can be expressed using these basic components.
- These components hide many of the implementation details that are not relevant to understanding the important differences among alternate architectures.

Two basic components, processors and schemas, play especially important roles in defining various architectures. The processors are application-independent software modules of a DBMS. Schemas are application-specific components that define database contents and structure. They are developed by the organizations to which the users belong. Users of a DBS include both persons performing ad hoc operations and application programs.

1.2 Processor Types in the Reference Architecture

Data management architectures differ in the types of processors present and the relationships among those processors. There are four types of processors, each performing different functions on data manipulation commands and accessed data: transforming processors, filtering processors, constructing processors, and accessing processors. Each of the processor types is discussed below.

1.2.1 Transforming Processor

Transforming processors translate commands from one language, called source

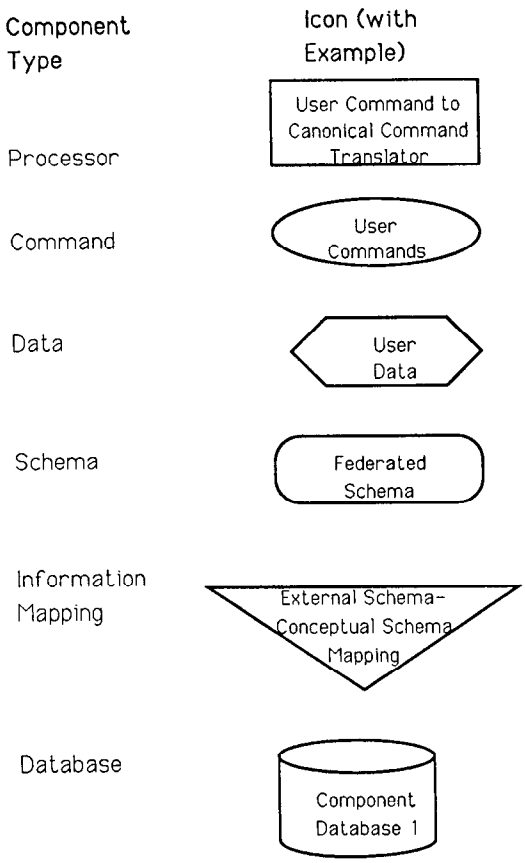

Figure 4. Basic system components of the data management reference architecture.

language, to another language, called target language, or transform data from one format (source format) to another format (target format). Transforming processors provide a type of data independence called *data model transparency* in which the data structures and commands used by one processor are hidden from other processors. Data model transparency hides the differences in query languages and data formats. For example, the data structures used by one processor can be modified to improve overall efficiency without requiring changes to other processors. Examples of command-transforming processors include the following:

- A command transformer that translates SQL commands into CODASYL data manipulation language commands [Onuegbe et al. 1983; Zaniolo 1979], allowing a CODASYL DBS to be processed using SQL commands.
- A program generator that translates SQL commands into equivalent COBOL programs allowing a file system to be processed using SQL commands.

For some command-transforming processors, there may exist companion data-transforming processors that convert data produced by the transformed commands into data compatible with the commands in the source format. For example, a data-transforming processor that is the companion to the above SQL-to-CODASYL command-transforming processor is a table builder that accepts individual database records produced by the CODASYL DBMS and builds complete tables for display to the SQL user.

Figure 5(a) illustrates a pair of companion transforming processors. Using information from schema A, schema B, and the mappings between them, the command-transforming processor converts commands expressed using schema A's description into commands expressed using schema B's description. Using the same information, the companion data-transforming processor transforms data described using schema B's description into data described using schema A's description.

To perform these transformations, a transforming processor needs mappings between the objects of each schema. The task of *schema translation* involves transforming a schema (schema A) describing data in one data model into an equivalent schema (schema B) describing the same data in a different data model. This task also generates the mappings that correlate the schema objects in one schema (schema B) to the schema objects in another schema (schema A). The task of *command transformation* entails using these mappings to translate commands involving the schema objects of one schema (schema B) into commands involving the schema objects of the other schema (schema A). The schema translation problem and the command transformation problem are further discussed in Sections 4.1 and 5.2, respectively.

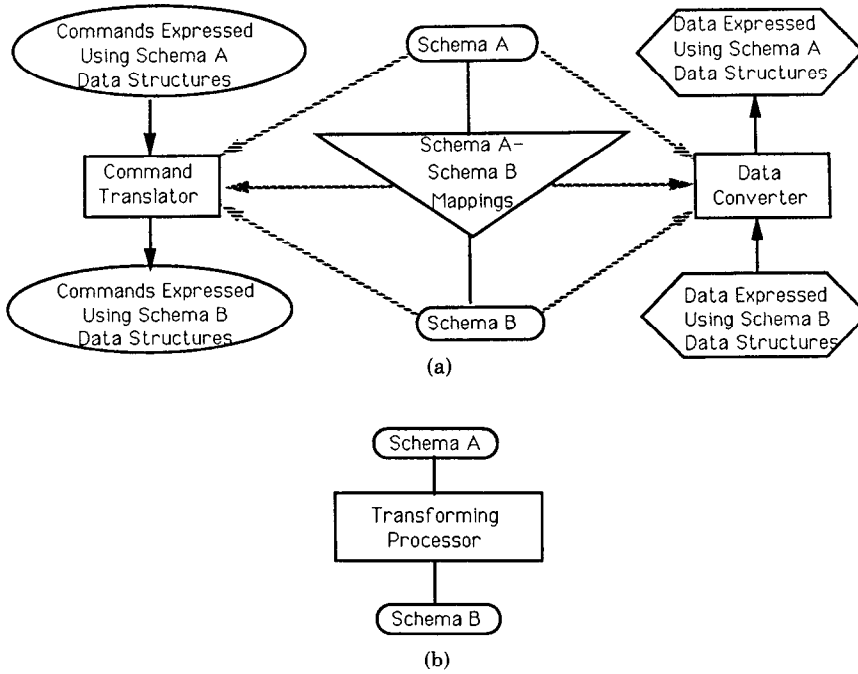

Figure 5. Transforming processors. (a) A pair of companion transforming processors. (b) An abstract transforming processor.

Mappings are associated with a transforming processor in one of two ways. In the first case, the mappings are encoded into the transforming processor's logic, making the transforming processor specific to the schemas. Alternatively, the mappings are stored in a separate data structure and accessed by the transforming processor when converting commands and data. This is a more general approach. It may also be possible to generate a transforming processor for transforming specific commands or data automatically. For example, an SQL-to-COBOL program generator might generate a specific data-transforming processor, the generated COBOL program, that converts data to the required form.

For the remainder of this paper we will illustrate a command-transforming processor and data converter pair as a single transforming processor as illustrated in Figure 4(b). This higher-level abstraction enables us to hide the differences between a single data-transforming processor, a single command-transforming processor, or a command-transforming processor and data converter pair.

1.2.2 Filtering Processor

Filtering processors constrain the commands and associated data that can be passed to another processor. Associated with each filtering processor are mappings that describe the constraints on commands and data. These constraints may either be embedded into the code of the filtering processor or be specified in a separate data structure. Examples of filtering processors include the following:

- Syntactic constraint checker, which checks commands to verify that they are syntactically correct.
- Semantic integrity constraint checker, which performs one or more of the following functions: (a) checks commands to verify that they will not violate semantic integrity constraints, (b) modifies commands in such a manner that when the

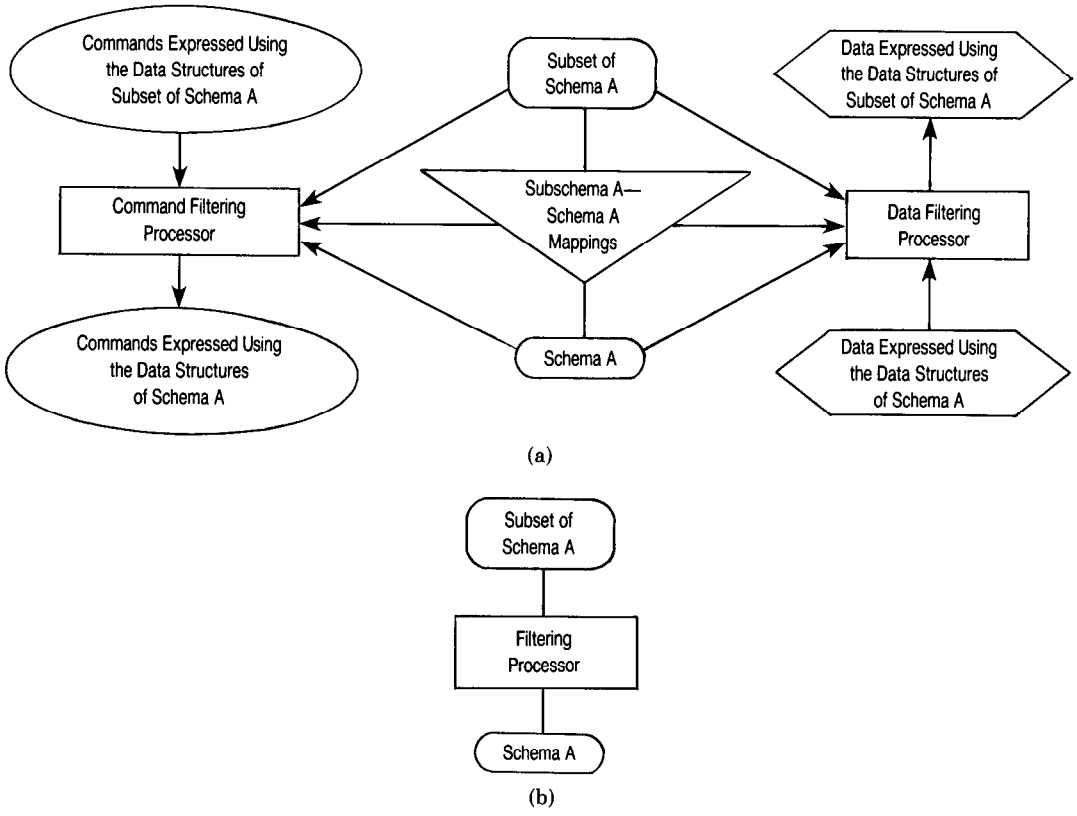

Figure 6. Filtering processors. (a) A pair of companion filtering processors. (b) An abstract filtering processor.

commands are interpreted, semantic integrity constraints will automatically be enforced, or (c) verifies that data produced by another processor does not violate any semantic integrity constraint.

- Access controller, which verifies that the user is permitted to perform the command on the indicated data or verifies that the user is permitted to use data produced by another processor.

Figure 6(a) illustrates two filtering processors, one that controls commands and one that controls data. Again, we will abstract command- and data-filtering processors into a single filtering processor as illustrated in Figure 6(b).

An important task that may be solved by a filtering processor is that of *view update*. This task occurs when the differences in data structures between the view and the schema is such that there may be more than one way to translate an update command. We do not discuss the view update task in more detail because we feel that a loosely coupled FDBS is not well suited to support updates, and solving this problem in a tightly coupled FDBS is very similar to solving it in a centralized or distributed DBMS [Sheth et al. 1988a].

1.2.3 Constructing Processor

Constructing processors partition and/or replicate an operation submitted by a single processor into operations that are accepted by two or more other processors. Constructing processors also merge data produced by several processors into a single data set for consumption by another single processor. They can support location, distribution, and replication transparencies because a processor submitting a command does not need to know the location, distribution, and

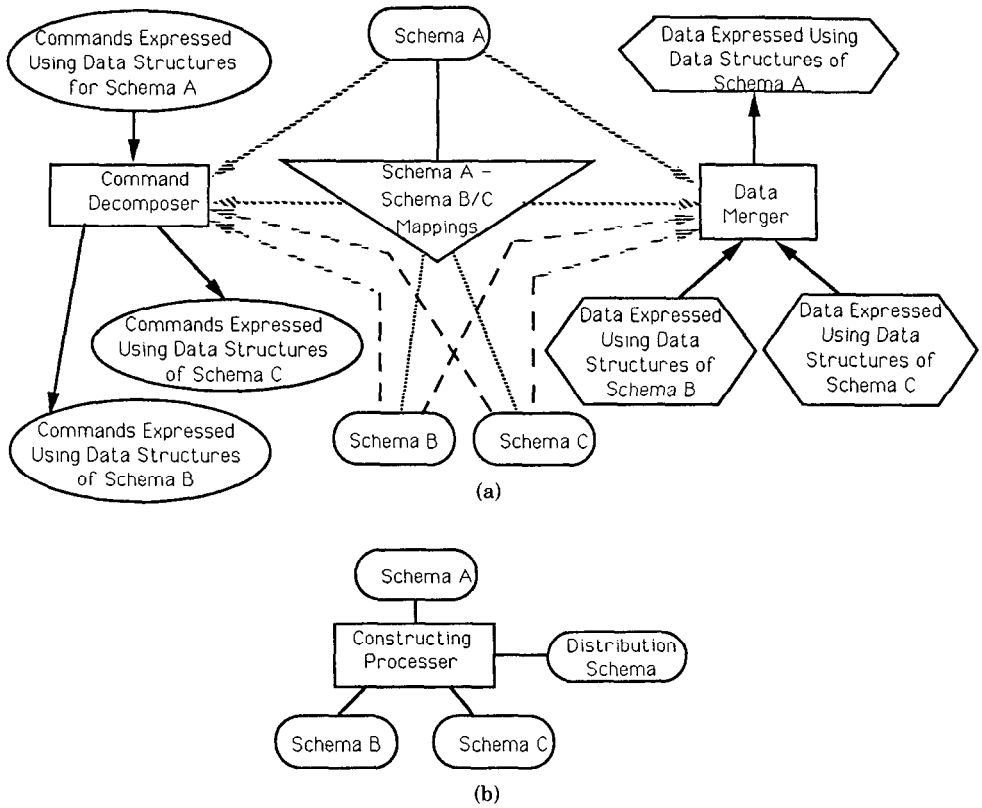

Figure 7. Constructing processors. (a) A pair of constructing processors. (b) An abstract constructing processor.

number of processors participating in processing that command.

Tasks that can be handled by constructing processors include the following:

- **Schema integration:** Integrating multiple schemes into a single schema
- **Negotiation:** Determining what protocol should be used among the owners of various schemas to be integrated in determining the contents of an integrated schema
- **Query (command) decomposition and optimization:** Decomposing and optimizing a query (command) expressed on an integrated schema
- **Global transaction management:** Performing the concurrency and atomicity control

These issues are further discussed in Sections 4 and 5. Figure 7(a) illustrates a pair of companion constructing processors. Using information from schema A, schema B, schema C, and the mappings from schema A to schemas B and C, the command decomposer uses the commands expressed using the schema A objects to generate the commands using the objects in schemas B and C. Schema A is an *integrated schema* that contains a description of all or parts of the data described by schemas B and C. Using the same information, the data merger generates data in the format of schema A objects from data in the formats of the objects in schemas B and C.

Again we will abstract the command partitioner and data merger pair into a single constructing processor as illustrated in Figure 7(b).

1.2.4 Accessing Processor

An *accessing processor* accepts commands and produces data by executing the

commands against a database. It may accept commands from several processors and interleave the processing of those commands. Examples of accessing processors include the following:

- A file management system that executes access procedures against stored file
- A special application program that accepts commands and generates data to be returned to the processor generating the commands
- A data manager of a DBMS containing data access methods
- A dictionary manager that manages access to dictionary data

Figure 8 illustrates an accessing processor that accepts data manipulation commands and uses access methods to retrieve data from the database.

Issues that are addressed by accessing processors include local concurrency control, commitment, backup, and recovery. These problems and their solutions are extensively discussed in the literature for centralized and distributed DBMSs. Some of the issues of adapting these problems to deal with heterogeneity and autonomy in the FDBSs are discussed in Section 5.4.

1.3 Schema Types in the Reference Architecture

In this section, we first review the standard three-level schema architecture for centralized DBMSs. We then extend it to a five-level architecture that addresses the requirements of dealing with distribution, autonomy, and heterogeneity in an FDBS.

1.3.1 ANSI/SPARC Three-Level Schema Architecture

The ANSI/X3/SPARC Study Group on Database Systems outlined a three-level data description architecture [Tsichritzis and Klug 1978]. The three levels of data description are the conceptual schema, the internal schema, and the external schema.

A *conceptual schema* describes the conceptual or logical data structures (i.e., the schema consists of objects that provide a conceptual- or logical-level description of the database) and the relationships among

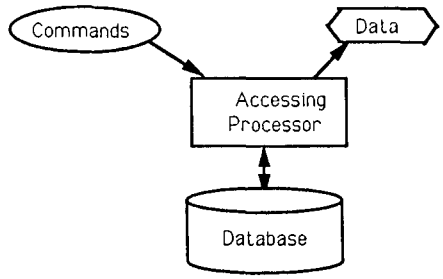

Figure 8. Accessing processor.

those structures. It is an attempt to describe all data of interest to an enterprise. In the context of the ANSI/X3/SPARC architecture, it is a database schema as expressed in the data definition language of a centralized DBMS. The *internal schema* describes physical characteristics of the logical data structures in the conceptual schema. These characteristics include information about the placement of records on physical storage devices, the placement and type of indexes and physical representation of relationships between logical records. Much of the description in the internal schema can be changed without having to change the conceptual schema. By making changes to the description in the internal schema and making the corresponding changes to the data in the database, it is possible to change the physical representation without changing any application program source code. Thus it is possible to fine tune the physical representation of data and optimize the performance of the DBMS in providing database access for selected applications.

Most users do not require access to all of the data in a database. Thus they do not require access to all of the schema objects in the conceptual schema. Each user or class of users may require access to only a portion of the database. The subset of the database that may be accessed by a user or a class of users is described by an *external schema.* Because different users may need access to different portions of the database, each user or a class of users may require a separate external schema.

In terms of the above constructs, filtering processors use the information in the external schemas to control what data can be

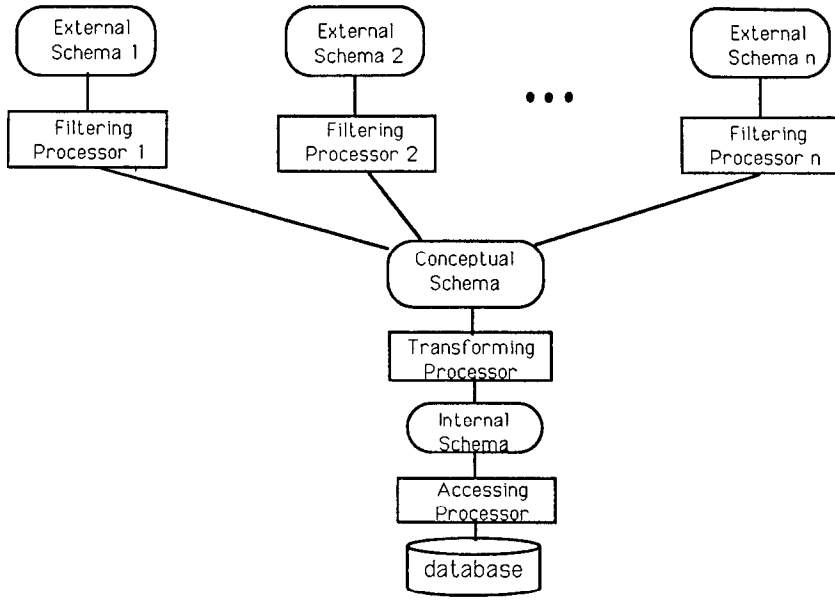

Figure 9. System architecture of a centralized DBMS.

accessed by which users. A transforming processor translates commands expressed using the conceptual schema objects into commands using the internal schema objects. An accessing processor executes the commands to retrieve data from the physical media. A system architecture consisting of both processors and schemas of a centralized DBS is shown in Figure 9.

1.3.2 A Five-Level Schema Architecture for Federated Databases

The three-level schema architecture is adequate for describing the architecture of a centralized DBMS. It, however, is inadequate for describing the architecture of an FDBS. The three-level schema must be extended to support the three dimensions of a federated database system—distribution, heterogeneity, and autonomy. Examples of extended schema architectures include a four-level schema architecture in Mermaid [Templeton et al. 1987b], five-level schema architectures in DDTS [Devor et al. 1982b] and SIRIUS-DELTA [Litwin et al. 1982], and others [Blakey 1987; Ram and Chastain 1989]. We have adapted these architectures for our five-level schema architecture for federated systems shown in Figure 10. A system architecture consisting of both processors and schemas of an FDBS is shown in Figure 11.

The five-level schema architecture of an FDBS includes the following:

Local Schema: A local schema is the conceptual schema of a component DBS. A local schema is expressed in the native data model of the component DBMS, and hence different local schemas may be expressed in different data models.

Component Schema: A component schema is derived by translating local schemas into a data model called the *canonical* or *common data model* (CDM) of the FDBS. Two reasons for defining component schemas in a CDM are (1) they describe the divergent local schemas using a single representation and (2) semantics that are missing in a local schema can be added to its component schema. Thus they facilitate negotiation and integration tasks performed when developing a tightly coupled FDBS. Similarly, they facilitate negotiation and specification of views and multidatabase queries in a loosely coupled FDBS.

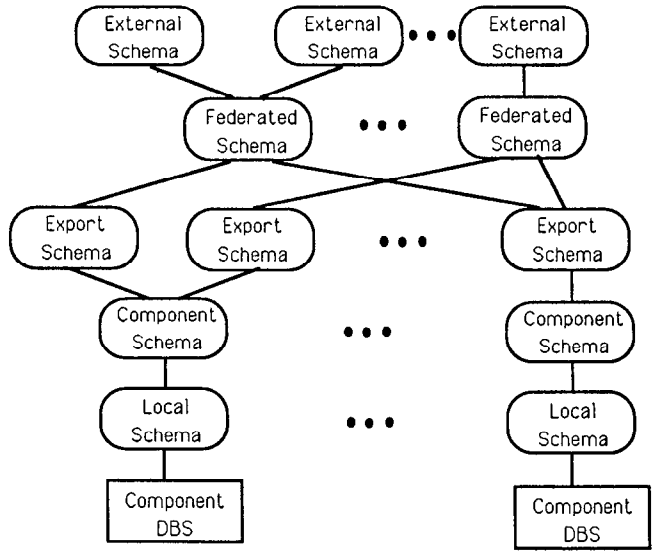

Figure 10. Five-level schema architecture of an FDBS.

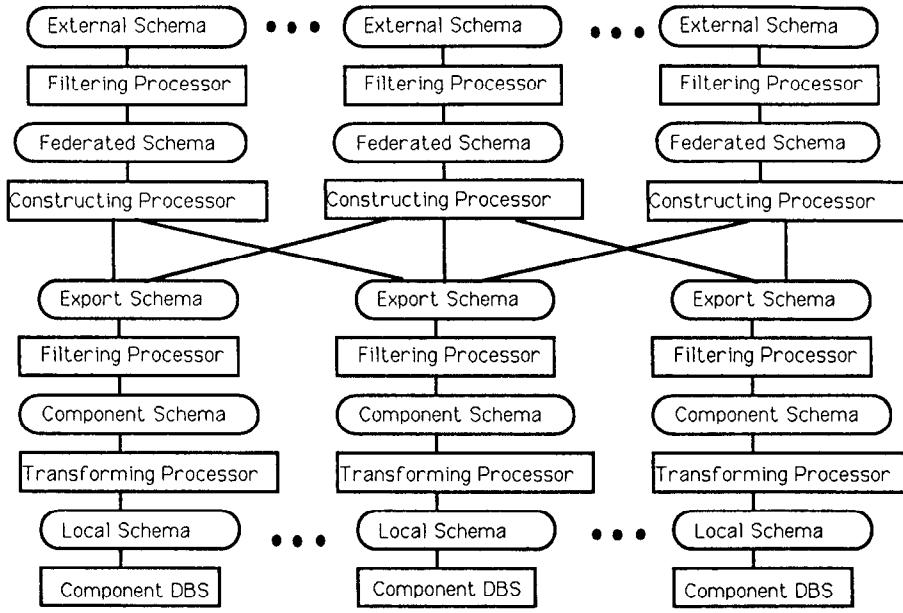

Figure 11. System architecture for an FDBS.

The process of schema translation from a local schema to a component schema generates the mappings between the component schema objects and the local schema objects. Transforming processors use these mappings to transform commands on a component schema into commands on the corresponding local schema.

Such transforming processors and the component schemas support the heterogeneity feature of an FDBS.

Export Schema: Not all data of a component DBS may be available to the federation and its users. An export schema represents a subset of a component schema that is available to the FDBS. It may include access control information regarding its use by specific federation users. The purpose of defining export schemas is to facilitate control and management of association autonomy. A filtering processor can be used to provide the access control as specified in an export schema by limiting the set of allowable operations that can be submitted on the corresponding component schema. Such filtering processors and the export schemas support the autonomy feature of an FDBS.

Alternatively, the data available to the FDBS can be defined as the transactions that can be executed by a component DBS (e.g., [Ge et al. 1987; Heimbigner and McLeod 1985; Veijalainen and Popescu-Zeletin 1988]). In this paper, however, we will not consider that case of exporting transactions.

Federated Schema: A federated schema is an integration of multiple export schemas. A federated schema also includes the information on data distribution that is generated when integrating export schemas. Some systems use a separate schema called a *distribution schema* or an *allocation schema* to contain this information. A constructing processor transforms commands on the federated schema into the commands on one or more export schemas. Constructing processors and the federated schemas support the distribution feature of an FDBS.

There may be multiple federated schemas in an FDBS, one for each class of federation users. A class of federation users is a group of users and/or applications performing a related set of activities. For example, in a corporate environment, all managers may be one class of federation users, and all employees and applications in the accounting department may be another class of federation users. A concept similar to that of federated schema is represented by the terms *import schema* [Heimbigner and McLeod 1985], *global schema* [Landers and Rosenberg 1982], *global conceptual schema* [Litwin et al. 1982], *unified schema*, and *enterprise schema*, although the terms other than import schemas are usually used when there is only one such schema in the system.

External Schema: An external schema defines a schema for a user and/or application or a class of users/applications. Reasons for the use of external schemas are as follows:

- **Customization:** A federated schema can be quite large, complex, and difficult to change. An external schema can be used to specify a subset of information in a federated schema that is relevant to the users of the external schema. They can be changed more readily to meet changing users' needs. The data model for an external schema may be different than that of the federated schema.
- **Additional integrity constraints:** Additional integrity constraints can also be specified in the external schema.
- **Access control:** Export schemas provide access control with respect to the data managed by the component databases. Similarly, external schemas provide access control with respect to the data managed by the FDBS.

A filtering process analyzes the commands on an external schema to ensure their conformance with access control and integrity constraints of the federated schema. If an external schema is in a different data model from that of the federated schema, a transforming processor is also needed to transform commands on the external schema into commands on the federated schema.

Most existing prototype FDBSs support only one data model for all the external schemas and one query language interface. Exceptions are a version of Mermaid that supported two query language interfaces, SQL and ARIEL, and a version of DDTS that supported SQL and GORDAS (a query language for an extended ER model).

Future systems are likely to provide more support for multimodel external schemas and multiquery language interfaces [Cardenas 1987; Kim 1989].

Besides adding to the levels in the schema architecture, heterogeneity and autonomy requirements may also dictate changes in the content of a schema. For example, if an FDBS has multiple heterogeneous DBMSs providing different data management capabilities, a component schema should contain information on the operations supported by a component DBMS.

An FDBS may be required to support local and external schemas expressed in different data models. To facilitate their design, integration, and maintenance, however, all component, export, and federated schemas should be in the same data model. This data model is called *canonical* or *common data model* (CDM). A language associated with the CDM is called an *internal command language*. All commands on federated, export, and component schemas are expressed using this internal command language.

Database design and integration is a complex process involving not only the structure of the data stored in the databases but also the semantics (i.e., the meaning and use) of the data. Thus it is desirable to use a high-level, semantic data model [Hull and King 1987; Peckham and Maryanski 1988] for the CDM. Using concepts from object-oriented programming along with a semantic data model may also be appropriate for use as a CDM [Kaul et al. 1990]. Although many existing FDBS prototypes use some form of the relational model as the CDM (Appendix), we believe that future systems are more likely to use a semantic data model or a combination of an object-oriented model and a semantic data model. Most of the semantic data models will adequately meet requirements of a CDM, and the choice of a particular one is likely to be subjective. Because a CDM using a semantic data model may provide richer semantic constructs than the data models used to express the local schemas, the component schema may contain more semantic information than the corresponding local schema. The additional semantics are supplied by the FDBS developer during the schema design, integration, and translation processes.

The five-level schema architecture presented above has several possible redundancies.

- **Redundancy between external and federated schemas:** External schemas can be considered redundant with federated schemas since a federated schema could be generated for every different federation user. This is the case in the schema architecture of Heimbigner and McLeod [1985] (they use the term *import schema* rather than federated schema). In loosely coupled FDBSs, a user defines the federated schema by integrating export schemas. Thus there is usually no need for an additional level. In tightly coupled FDBSs, however, it may be desirable to generate a few federated schemas for widely different classes of users and to customize these further by defining external schemas. Such external schemas can also provide additional access control.
- **Redundancy between an external schema of a component DBS and an export schema:** If a component DBMS supports proper access control security features for its external schemas and if translating a local schema into a component schema is not required (e.g., the data model of the component DBMS is the same as CDM of the FDBS), then the external schemas of a component DBS may be used as an export schema in the five-level schema architecture (external schemas of component DBSs are not shown in the five-level schema architecture of Figure 10).
- **Redundancy between component schemas and local schemas:** When component DBSs uses CDM of the FDBS and have the same functionality, it is unnecessary to define component schemas.

Figure 12 shows an example in which some of the schema levels are not used. No external schemas are defined over Federated Schema 2 (all of it is presented to all

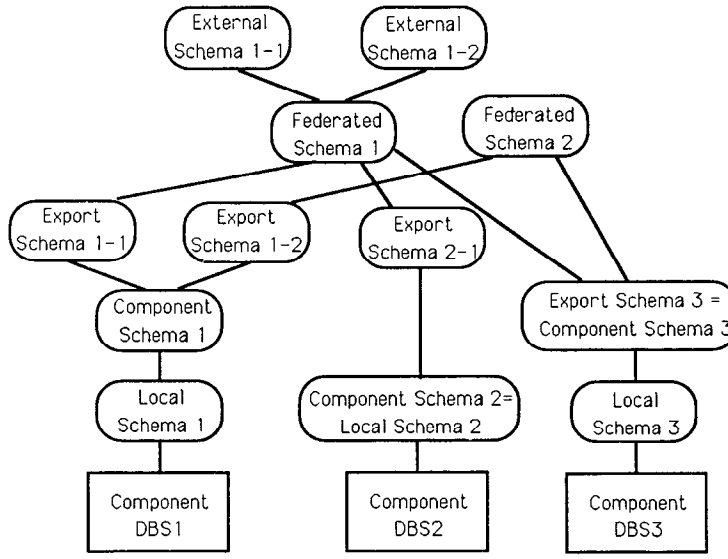

Figure 12. Example FDBS schemas with missing schemas at some levels.

federation users using it). Component Schema 2 is the same as the Local Schema 2 (the data model of the Component DBMS 2 is the same as the CDM). No export schema is defined over Component Schema 3 (all of it is exported to the FDBS).

An important type of information associated with all FDBS schemas is the mappings. These correlate schema objects at one level with the schema objects at the next lower level of the architecture. Thus, there are mappings from each external schema to the federated schema over which it is defined. Similarly, there are mappings from each federated schema to all of the export schemas that define it. The mappings may either be stored as a part of the schema information or as distinct objects within the FDBS data dictionary (which also stores schemas). The amount of dictionary information needed to describe a schema object in one type of schema may be different from that needed for another type of schema. For example, the description of an entity type in a federated schema may include the names of the users that can access it, whereas such information is not stored for an entity type in a component schema. The types of schema objects in one type of schema may also vary from those in another type of schema. For example, a federated schema may have schema objects describing the capabilities of the various component DBMSs in the system, whereas no such objects exist in the local schemas.

Two important features of the schema architecture are how autonomy is preserved and how access control is managed. These involve exercising control over schemas at different levels. Two types of administrative individuals are involved in developing, controlling, and managing an FDBS:

- *A component DBS administrator* (*component DBA*) manages a component DBS. There is one component DBA[5] for each component DBS. The local, component, and export schemas are controlled by the component DBAs of the respective component DBSs. A key management function of a component DBA

[5] Here a database administrator is a *logical* entity. In reality, multiple authorized individuals may play the role of a single (logical) DBA, or the same individual may play the role of the component DBA for multiple component DBSs.

is to define the export schemas that specify the access rights of federation users to access different data in the component databases.

- *A federation DBA* defines and manages a federated schema and the external schemas related to the federated schema. There can be one federation DBA for each federated schema or one federation DBA for the entire FDBS. Each federation DBA in a tightly coupled FDBS is a specially authorized system administrator and is not a federation user. In a loosely coupled FDBS, federated schemas are defined and maintained by the users, not by the system-assigned federation DBA. This is further discussed in Section 2.1.

2. SPECIFIC FEDERATED DATABASE SYSTEM ARCHITECTURES

The architecture of an FDBS is primarily determined by which schemas are present, how they are arranged, and how they are constructed. In this section, we begin by discussing the loosely coupled and tightly coupled architectures of our taxonomy in additional detail. Then we discuss how several alternate architectures can be derived from the five-level schema architecture by inserting additional basic components, removing all basic components of a specific type, and arranging the components of the five-level schema architecture in different ways. We then discuss assignment of components to computers. Finally, we briefly discuss four case studies.

2.1 Loosely Coupled and Tightly Coupled FDBSs

With the background of Section 1, we discuss distinctions between the loosely coupled and tightly coupled FDBSs in more detail.

2.1.1 Creation and Administration of Federated Schemas

The process of creating a federated schema takes different forms. In a loosely coupled FDBS, it typically takes the form of schema importation (e.g., defining "import schemas" in Heimbigner and McLeod [1985]), defining a view using a set of operators (e.g., defining "superviews" in Motro and Buneman [1981]), or defining a view using a query in a multidatabase language ([Czejdo et al. 1987; Litwin and Abdellatif 1986]; see Section 5.1). In a tightly coupled FDBS, it takes the form of schema integration ([Batini et al. 1986]; see Section 4.4).

A typical process of developing federated schemas in a loosely coupled FDBS is as follows. Each federation user is the administrator of his or her own federated schema. First, a federation user looks at the available set of export schemas to determine which ones describe data he or she would like to access. Next, the federation user defines a federated schema by importing the export schema objects by using a user interface or an application program or by defining a multidatabase language query that references export schema objects. The user is responsible for understanding the semantics of the objects in the export schemas and resolving the DBMS and semantic heterogeneity. In some cases, component DBMS dictionaries and/or the federated DBMS dictionary may be consulted for additional information. Finally, the federated schema is named and stored under account of the federation user who is its owner. It can be referenced or deleted at any time by that federation user.

A typical scenario for the administration of a tightly coupled FDBS is as follows. For simplicity, we assume single (logical) federation DBA for the entire tightly coupled FDBS. Export schemas are created by negotiation between a component DBA and the federation DBA; the component DBA has authority or control over what is included in the export schemas. The federation DBA is usually allowed to read the component schemas to help determine what data are available and where they are located and then negotiate for their access. The federation DBA creates and controls the federated schemas. External schemas are created by negotiation between a federation user (or a class of federation users) and the federation DBA who has the authority over what is included in each

external schema. It may be possible to institute detailed and well-defined negotiation protocols as well as business rules (or some types of constraints) for creating, modifying, and maintaining the federated schemas.

Based on how often the federated schemas are created and maintained as well as on their stability, an FDBS may be termed *dynamic* or *static*. Properties of a dynamic FDBS are as follows: (a) A federated schema can be promptly created and dropped; (b) there is no predetermined process for controlling the creation of a federated schema. As described above, defining a federated schema in a loosely coupled FDBS is like creating a view over the schemas of the component DBSs. Since such a federated schema may be managed *on the fly* (created, changed, dropped easily) by a user, loosely coupled FDBSs are dynamic. A tightly coupled federation is almost always static because creating a federated schema is like database schema integration. A federated schema in a tightly coupled FDBS evolves gradually and in a more controlled fashion.

2.1.2 Case for Loosely Coupled FDBS

A loosely coupled FDBS provides an interface to deal with multiple component DBMSs directly. A typical way to formulate queries is to use a multidatabase language (see Section 5.1). This architecture has the following advantages:

- A user can precisely specify relationships and mappings among objects in the export schema. This is desirable when the federation DBA is unable to specify the mappings in order to integrate data in multiple databases in a manner meaningful to the user's precise needs [Litwin and Abdellatif 1986].
- It is also possible to support multiple semantics since different users can import or integrate export schemas differently and maintain different mappings from their federated schemas to export schemas. This can be a significant advantage when the needs of the federation users cannot be anticipated by the federation DBA [Litwin and Abdellatif 1986].

An example of multiple semantics is as follows. Suppose that there are two export schemas, each containing the entity SHOE. The colors of SHOE in one component schema, *schema1*, are brown, tan, cream, white, and black. The colors of SHOE in the other component schema, *schema2*, are brown, tan, white, and black. Users defining different federated schemas may define different mappings that are relevant to their applications. For example,

- *User1* maps cream in his federated schemas to cream in *schema1* and tan in *schema2*,
- *User2* maps cream in her federated schema to tan or cream in *schema1* and tan or white in *schema2*.

Proponents of the loosely coupled architecture argue that a federated schema created and maintained by a single federation DBA is utopian and totalitarian in nature [Litwin 1987; Rusinkiewicz 1987]. We feel that a loosely coupled approach may be better suited for integrating a large number of very autonomous read only databases accessible over communication networks (e.g., public databases of the types discussed by Litwin and Abdellatif [1986]). User management of federated schemas means that the FDBMS can do little to optimize queries. In most cases, however, the users are free to use their own understanding of the component DBSs to design a federated schema and to specify queries to achieve good performance.

2.1.3 Case for Tightly Coupled FDBS

The loosely coupled approach may be ill suited for more traditional business or corporate databases, where system control (via DBAs that represent local and federation level authories) is desirable, where the users are naive and would find it difficult to perform negotiation and integration themselves, or where location, distribution, and replication transparencies are desirable. Furthermore, in our opinion, a loosely

coupled FDBS is not suitable for update operations. Updating in a loosely coupled FDBS may degrade data integrity. When a user of a loosely coupled FDBSs creates a federated schema using a view definition process, view update transformations are often not determined. The users may not have complete information on the component DBSs and different users may use different semantic interpretations of the data managed by the component DBSs (i.e., loosely coupled FDBSs support multiple semantic interpretations). Thus different users can define different federated schemas over the same component DBSs, and different transformations may be chosen for the same updates submitted on different federated schemas. Similar problems can occur in a tightly coupled FDBS with multiple federations but can be resolved at the time of federated schema creation through schema integration. A federation DBA creating a federated schema using a schema integration process can be expected to have more complete knowledge of the component DBSs and other federated schemas. In addition to the update transformation issue, transaction management issues need to be addressed (see Section 5.4).

A tightly coupled FDBS provides location, replication, and distribution transparency. This is accomplished by developing a federated schema that integrates multiple export schemas. The transparencies are managed by the mappings between the federated schema and the export schemas, and a federation user can query using a classical query language against the federated schema with an illusion that he or she is accessing a single system. A loosely coupled system usually provides none of these transparencies. Hence a user of a loosely coupled FDBS has to be sophisticated to find appropriate export schemas that can provide required data and to define mappings between his or her federated schema and export schemas. Lack of adequate semantics in the component schemas make this task particularly difficult. Let us now discuss two alternatives for tightly coupled FDBSs in more detail.

In a tightly coupled FDBS with a single federation, all export schemas are integrated to develop a single federated schema. Sometimes an organization will insist on having a single federated schema (also called enterprise schema or global conceptual schema) to have a single point of control for all data sharing in the organization across the component DBS boundaries. Using a single federated schema helps in defining uniform semantics of the data in the FDBS. With a single federated schema, it is also easier to enforce constraints that cross export schemas (and hence multiple databases) then when multiple federated schemas are allowed.

Because one federated schema is created by integrating all export schemas and because this federated schema supports data requirements of all federation users, it may become too large and hence difficult to create and maintain. In this case, it may become necessary to support external schemas for different federation users.

A tightly coupled FDBS with multiple federations allows the tailoring of the use of the FDBS with respect to multiple classes of federation users with different data access requirements. Integrations of the same set of schemas can lead to different integrated schemas if different semantics are used. Thus this architecture can support multiple semantics, but the semantics are decided upon by the federation DBAs when defining the federated schemas and their mappings to the export schemas. A federation user can select from among multiple alternative mappings by selecting from among multiple federated schemas. When an FDBS allows updates, multiple semantics could lead to inconsistencies. For this reason, federation DBAs have to be very careful in developing the federated schemas and their mappings to the export schemas. Updates are easier to support in tightly coupled FDBSs where DBAs carefully define mappings than in a loosely coupled FDBS where the users define the mappings.

2.2 Alternative FDBS Architectures

In this section, we discuss how processors and schemas are combined to create various FDBS architectures.

2.2.1 A Complete Architecture of a Tightly Coupled FDBS

An architecture of a tightly coupled FDBS, shown in Figure 11, consists of multiple basic components as described below.

- **Multiple export schemas and filtering processors:** Any number of external schemas can be defined, each with its own filtering processor. Each external schema supports the data requirements of a single federation user or a class of federation users.
- **Multiple federated schemas and constructing processors:** Any number of federated schemas can be defined, each with its own constructing processor. Each federated schema may integrate different export schemas (and the same export schema may be integrated differently in different federated schemas).
- **Multiple export schemas and filtering processors:** Multiple export schemas represent different parts of a database to be integrated into different federated schemas. A filtering processor associated with an export schema supports access control for the related component schema.
- **Multiple component schemas and transforming processors:** Each component schema represents a different component database expressed in the CDM. Each transforming processor transforms a command expressed on the associated component schema into one or more commands on the corresponding local schema.

2.2.2 Architectures with Missing Basic Components

There are several architectures in which all of the processors of one type and all schemas of one type are missing. Several examples follow.

- **No transforming processors or component schemas:** All of the local schemas are described in a single data model. In other words, the FDBS does not support component DBSs that use different data models. Hence there is no need for component schemas. Mermaid [Templeton et al. 1987b] falls into this category.[6]
- **No filtering processors or export schemas:** All of the component schemas are integrated into a single federated schema resulting in a tightly coupled system in which component DBAs do not control what users can access. This architecture fails to support component DBS autonomy fully. UNIBASE [Brzezinski et al. 1984] is in this category, and hence it is classified as a nonfederated system.
- **No constructing processor:** The user or programmer performs the constructing process via a query or application program containing references to multiple export schemas. The programmer must be aware of what data are available in each export schema and whether data are replicated at multiple sites. This architecture, classified as a loosely coupled FDBS, fails to support location, distribution, and replication transparencies. If data are copied or moved between component databases, any query or application using them must be modified.

In practice, two processors may be combined into a single module, or two schemas may be combined into a single implementation schema. For example, a component schema and its export schemas are frequently combined into a single schema with a single processor that performs both transformation and filtering.

2.2.3 Architectures with Additional Basic Components

There are several types of architectures with additional components that are extensions or variations of the basic components of the reference architecture. Such components enhance the capabilities of an FDBS. Examples of such components include the following:

- **Auxiliary schema:** Some FDBSs have an additional schema called an auxiliary

[6] Its design, however, has provisions to store model transformation information and attach a transforming processor.

schema that stores the following types of information:

- Data needed by federation users but not available in any of the (preexisting) component DBSs.
- Information needed to resolve incompatibilities (e.g., unit translation tables, format conversion information).
- Statistical information helpful in performing query processing and optimization.

Multibase [Landers and Rosenberg 1982] describes the first two types of information in its auxiliary schema, whereas DQS [Belcastro et al. 1988] describes the last two types of information in its auxiliary schema. Mermaid [Templeton et al. 1987b] describes the third type of information in its federated schema. As illustrated in Figure 13, the auxiliary schema and the federated schema are used by constructing processors. It is also possible to consider the auxiliary schema to be a part (or subschema) of a federated schema.

- **Enforcing constraints among component schemas:** As illustrated in Figure 14, an FDBS can have a filtering processor in addition to a constructing processor between a federated schema and the component schemas. The filtering processor enforces constraints that span multiple component schemas. The constructing processor, as discussed before, transforms a query into subqueries against the component schemas of the component DBSs. Integrity constraints may be stored in an external schema or a federated schema. The constraints may involve data represented in multiple export schemas. The filtering processor checks and modifies each update request so when data in multiple component databases are modified, the intercomponent constraints are not violated. This capability is appropriate in a tightly coupled system in which constraints among multiple component databases must be enforced. An early description of DDTS [Devor et al. 1982a] suggested enforcement of semantic integrity constraints spanning components in this manner.

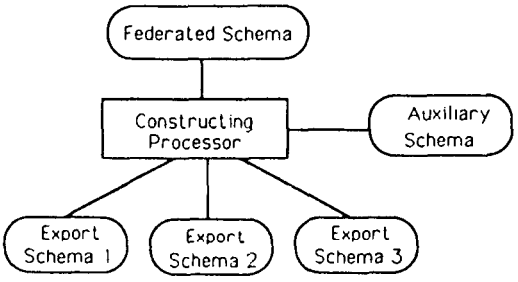

Figure 13. Using an auxiliary schema to store translation information needed by a constructing processor.

This, however, can limit or conflict with the autonomy of the component DBSs.

2.2.4 *Extended Federated Architectures*

To allow a federation user to access data from systems other than the component DBSs, the five-level schema architecture can be extended in additional ways.

- **Atypical component DBMS:** Instead of a typical centralized DBMS, a component DBMS may be a different type of data management system such as a file server, a database machine, a distributed DBMS, or an FDBMS. OMNIBASE uses a distributed DBMS as one of its component DBMSs [Rusinkiewicz et al. 1989]. Figure 15 illustrates how one FDBS can act as a backend for another FDBS. By making local schema A2 of FDBS A the same as external schema B2 of FDBS B, the component DBS A2 of FDBS A is replaced by FDBS B.
- **Replacing a component database by a collection of application programs:** It is conceptually possible to replace some database tables by application programs. For example, a table containing pairs of equivalent Fahrenheit and Celsius values can be replaced by a procedure that calculates values on one scale given values on the other. A collection of conversion procedures can be modeled by the federated system as a special-component database. A special-access processor can be developed that accepts requests for conversion information and invokes the appropriate procedure rather

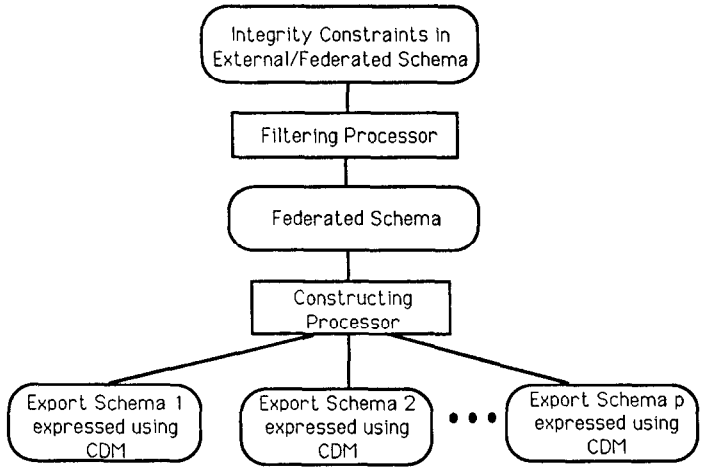

Figure 14. Using a filtering processor to enforce constraints across export schemas.

than access a stored database. Navathe et al. [1989] discuss a federated architecture being developed to provide access to databases as well as application programs.

2.3 Allocating Processors and Schemas to Computers

It is possible to allocate all processors and schemas to a single computer, perhaps to allow federation users to access data managed by multiple component DBSs on that computer. Usually, however, different component DBSs reside on different computers connected by a communication system. Different allocations of the FDBS components result in different FDBS configurations.

Figure 16 illustrates the configuration of a typical FDBS. A general-purpose computer at site 1 supports a single component DBS and two federation schemas for two different classes of federation users. Site 2 is a workstation that supports two export schemas, each containing different data for use by different federation users. Site 3 is a small workstation that supports a single federation user and no component DBS. Site 4 is a database computer that has one component DBS but supports no federation users.

It may be desirable to group a related set of processors and schemas into modules of larger granularity and allocate them as desired. For example, DDTS [Dwyer and Larson 1987] defines two types of modules: Application Processor and Data Processor (Figure 17). An *Application Processor* includes a federated schema with the associated constructing processor and all the external schemas defined over the federated schema with the associated filtering processors and transforming processors (if present). A *Data Processor* includes a local schema, a component schema, and the associated transforming processor and all export schemas defined over the component schema with associated filtering processors. An Application Processor performs the user interface and distributed transaction management and coordination functions and is located at every site at which there are federation users. A Data Processor performs the data management functions related to the data managed by a single component DBS and is located at every site at which a component DBS is located. A site can have either or both of the two modules. Mermaid [Templeton et al. 1987b] divides the processors and the schemas into four types of modules of smaller granularity.

Special communication processors can also be placed on each computer to enable processors on two different sites to communicate with each other. Communication

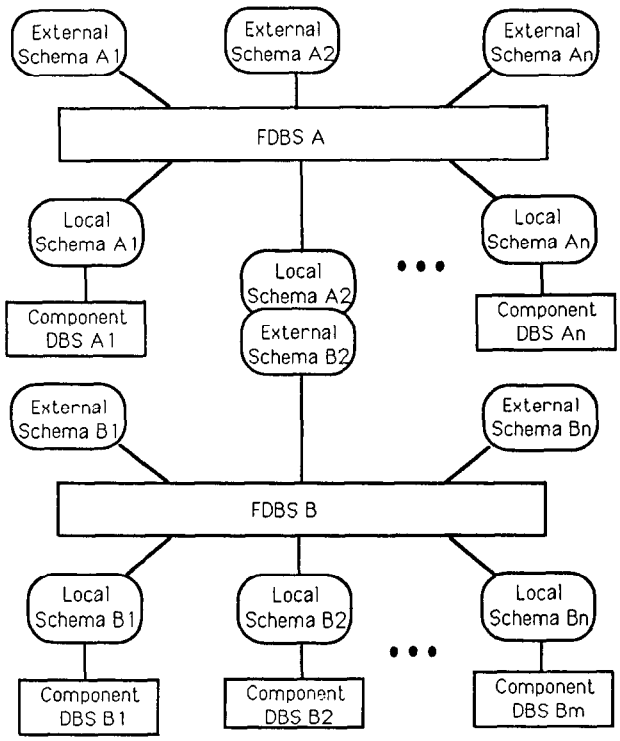

Figure 15. FDBS B acting as a back end to FDBS A.

processors are not shown in our reference architecture. They are placed between any pair of adjacent processors that are allocated to different computers.

2.4 Case Studies

In this section we relate the terms and concepts of the reference architecture to those used in four example FDBSs. Our purpose is not to survey these systems [Thomas et al. 1990] but to show how the reference architecture can be used to represent the architectures of various FDBSs uniformly. This uniform representation can greatly simplify the task of studying and comparing these systems.

2.4.1 DDTS

Figure 17 illustrates the original architecture of DDTS [Devor et al. 1982a] using the terminology of the reference architecture (to the left of each colon) and the terminology used by DDTS (to the right of each colon and in italics). It has a single federated schema called the Global Representation Schema, which is expressed in the relational data model. It has an external schema called the Conceptual Schema represented in the Entity-Category-Relationship (ECR) model [Elmasri et al. 1985]. Users formulate requests directly against the Conceptual Schema in the GORDAS query language [Elmasri 1981]. The ECR data model is rich in semantics (e.g., it shows cardinality and operation constraints on an entity's participation in relationships). The transforming part of the Translation and Integrity Control processor is responsible for translating requests written in GORDAS on the ECR data model into the internal form of a relational query language against the Global Representational Schema. The filtering part of the Translation and Integrity Control processor is responsible for modifying each query, so when it is processed, the constraints specified in the Conceptual Schema will be enforced. For example, a GORDAS query that deletes a record will

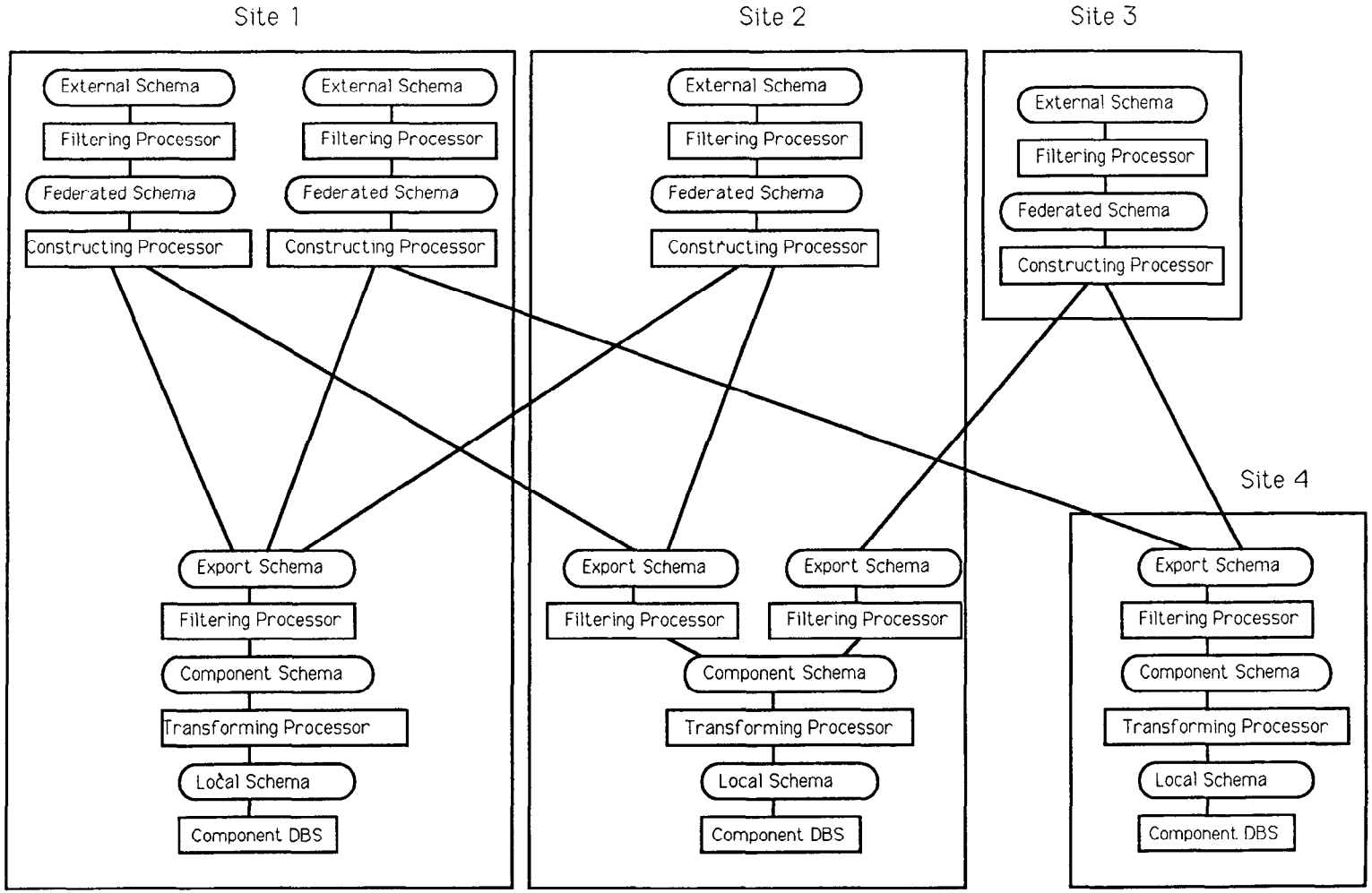

Figure 16. Typical FDBS system configuration.

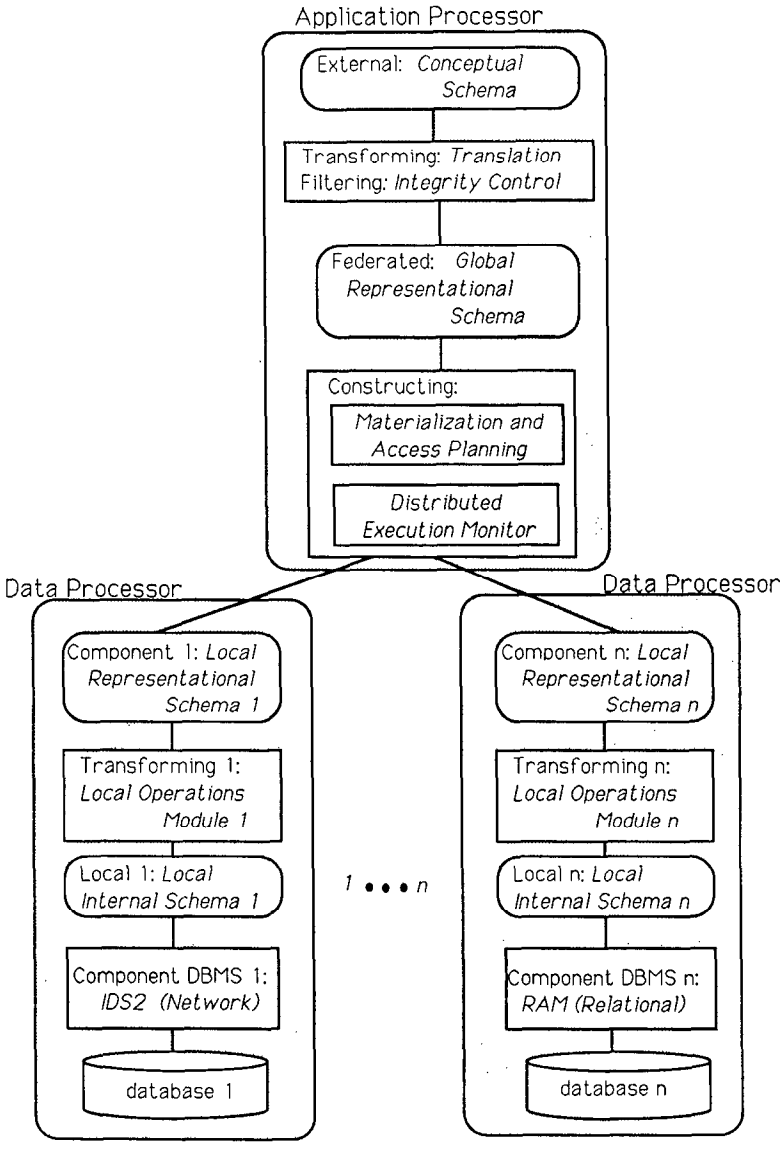

Figure 17. DDTS architecture.

be modified so that it verifies that deleting the record will not violate any semantic integrity constraints before the record is actually deleted. DDTS has since been extended to support external schemas [Dwyer and Larson 1987] expressed in the relational data model defined over the Global Representation Model. SQL is used to query such external schemas.

There are two separate processors in DDTS's constructing processor, reflecting the decision to separate distributed query optimization from distributed query execution. The Materialization and Access Planning component generates a distributed execution strategy consisting of sets of commands, each expressed in terms of one of the Local Representational

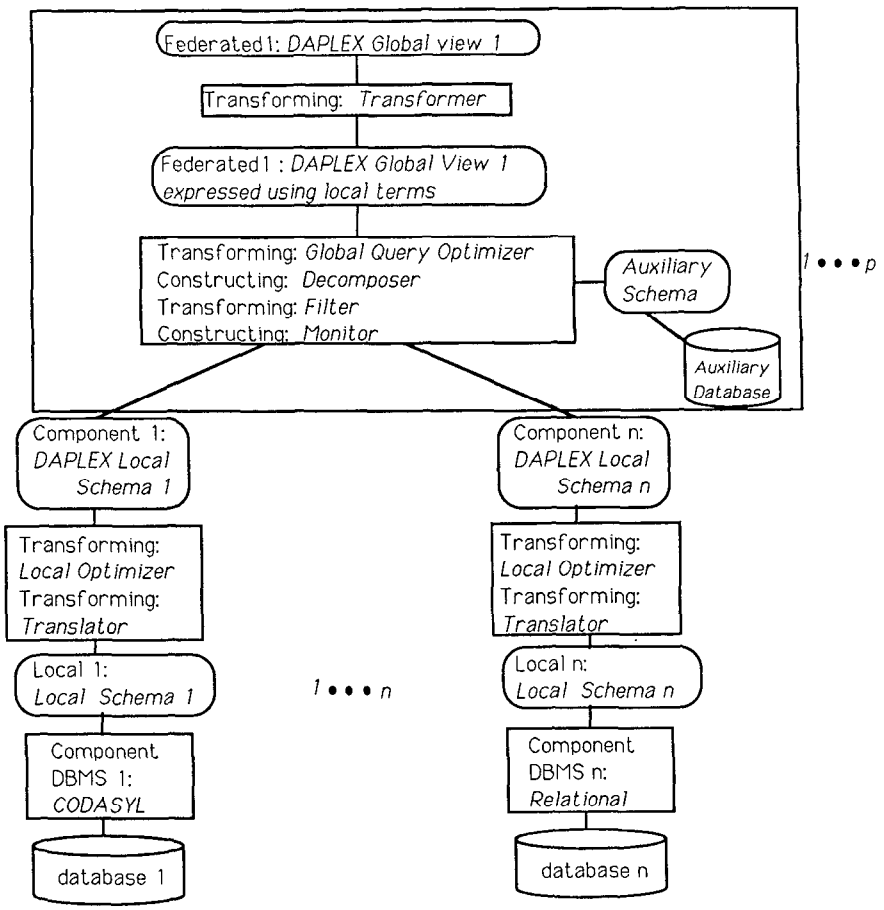

Figure 18. Multibase architecture.

(component) schemas. The distributed execution strategy can be saved for future execution or can be passed immediately to the Distributed Execution Monitor. The Distributed Execution Monitor is responsible for executing the distributed execution strategy, coordinating the execution of the sets of commands, and returning the results to the user.

The Local Operations (transforming) Module accepts a set of commands and transforms it into a form that can be executed by the component DBMS. Originally there were two CODASYL component DBMSs. Later a third component DBMS using the relational data model was added.

2.4.2 *Multibase*

Figure 18 illustrates the architecture of Multibase, again using the terminology of the reference architecture and the terminology used by Landers and Rosenberg [1982]. The federated schema is expressed in a functional data model called DAPLEX. The Transformer modifies global queries by inserting references to Local and Auxiliary schemas. The modified global query is then processed by a series of processors that generates sequences of DAPLEX single-site queries. These processors include:

- A Global Query Optimizer that produces a global plan,

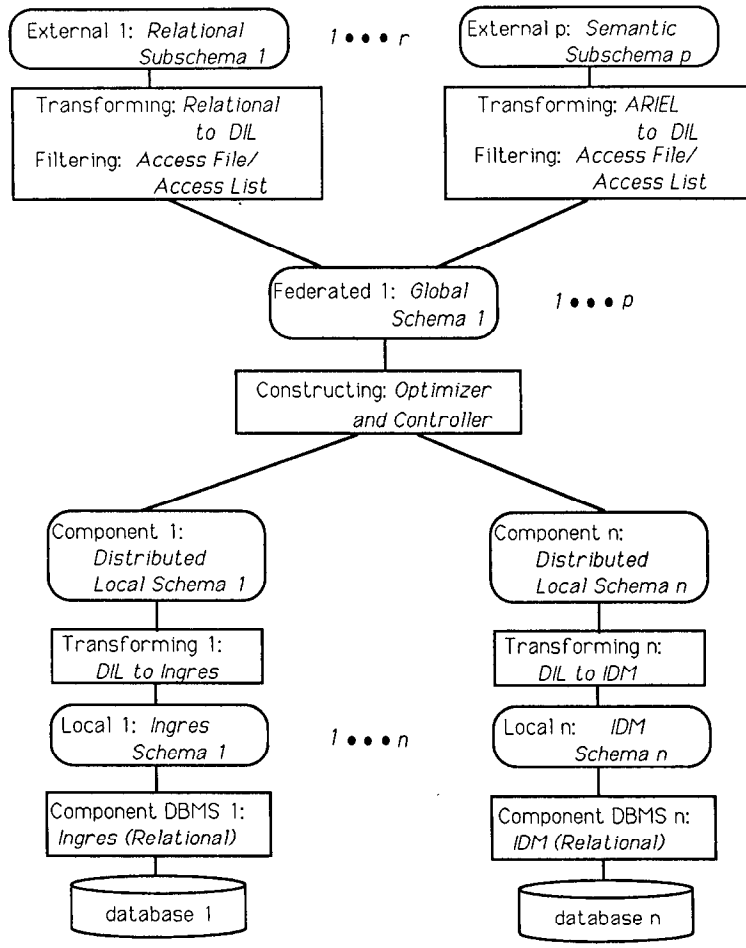

Figure 19. Mermaid architecture.

- A Decomposer that decomposes the global plan into single-site DAPLEX queries,
- A Filter that reduces the decomposed queries by removing operations from them that are not supported by the corresponding component DBMSs, and
- A Monitor that controls the distributed execution of the subqueries.

The Optimizer, Decomposer, and Filter may be cyclically invoked for nested global queries. Two types of transformations are performed on each DAPLEX single-site query:

- A Local Optimizer determines the optimal query-processing strategy for the single-site DAPLEX queries.
- A Translator converts the DAPLEX query to a form the component DBMS can process.

An Auxiliary schema holds additional data not stored in any component DBMS and information needed to resolve inconsistencies. Component DBMSs supported by Multibase include both CODASYL and relational.

2.4.3 Mermaid

Figure 19 illustrates the architecture of Mermaid, again using the terminology of the reference architecture and the terminology used by Templeton et al. [1987b]. Users may formulate requests using either SQL on a relational schema or ARIEL, a

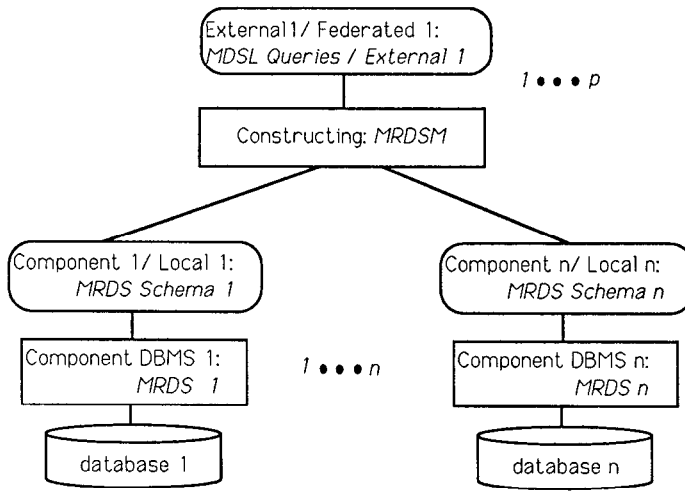

Figure 20. MRDSM architecture.

user-friendly query language developed at Systems Development Corporation (now Unisys), on what is called a Semantic Subschema. The federated schema is called a Global Schema and is expressed in the relational model. User requests are transformed to the internal command language called the Distributed Intermediate Language (DIL). DIL requests are processed by a constructing processor that produces and executes a query plan. A transforming processor (one for each component DBS) translates DIL requests into the query language of the component DBMS, interacts with the component DBMS, and sends data to other transforming processors (e.g., for joins and unions). Component DBMSs include commercial relational DBMSs on Sun® workstations and a database machine with a minicomputer host [Thomas et al. 1990].

2.4.4 MRDSM

MRDSM is a loosely coupled FDBS (called a *multidatabase system* by its originators) in which a programmer/user formulates a request involving data from component DBSs. The system provides a multidatabase language called MDSL [Litwin and Abdellatif 1987] to facilitate formulating such queries. Figure 20 illustrates the architecture of MRDSM using the terminology of the reference architecture and the terminology of Litwin [1985].

3. FEDERATED DATABASE SYSTEM EVOLUTION PROCESS

There are two approaches to managing distributed data: installing a distributed DBMS or adding a layer of software above existing DBMSs to create an FDBS system. In the first approach, installing a distributed DBMS requires (1) changes and disruption of the existing applications because they do not have the dichotomy of local versus global operations, (2) a complete change of the organizational structure for information management because this approach does not respect the autonomy of existing DBSs, and (3) replacement of the existing centralized DBMSs by a distributed DBMS.

The federation approach offers a preferred evolutionary path. It allows continued operation of existing applications to remain unchanged, preserves most of the organizational structure, supports controlled integration of existing databases, and facilitates incorporation of new applications and new databases. Although

® Sun is a trademark of Sun Microsystems, Inc.

existing applications need not be changed in an FDBS, as the old applications are modified, the component databases may be standardized, and redundant data (unless required for improving availability or access time) may be removed. New applications may be coded using an external schema defined on a federated schema.

An FDBS evolves through gradual integration of a set of interrelated DBSs. It evolves as the new component databases are added and the existing ones are modified. This evolution process can be divided into three phases: preintegration, developing a federated database system, and federated database system operation. These phases (or the activities within the phases) need not follow serially from one phase to the next; each phase may be performed several times, and previous phases may be revisited and their results revised.

The *preintegration phase* deals with the situation in which data reside in files and are not managed by any DBMS yet need to be accessed by federation users. Two general approaches are possible:

- **Migrate the files to a DBMS:** Files can be migrated to a DBMS by performing three activities: (1) develop a component schema that describes the data in the files, (2) load the DBMS with data from the files, and (3) modify existing application programs to access the DBMS rather than access the files directly.
- **Extend the file system to support DBMS-like features:** By extending the file system to support DBMS-like features, a file system can be treated as a component DBMS. In this approach, the following activities are performed: (1) create or generate a component schema that describes data in the files, (2) create backup and recovery facilities in the file system if the federation users will perform updates to data in the file system, and (3) create appropriate filtering and transforming processors that will convert commands expressed in the FDBS's internal language to the commands that can be processed by the file system.

Both migration and extension of file systems are expensive. Many developers of the FDBSs elect to use migration because they do not have access to the source code of the file system and because they want their data maintained by a commercially available DBMS. Since existing application programs are affected, however, this activity is very difficult and often organizationally very sensitive.

The second phase of the FDBS evolution, *developing a federated database system*, involves creating the component, export, federated, and external schemas, defining the mappings between various schemas, and implementing the associated processors. This is a critical task for the success of an FDBS. A methodology that can be used to manage this phase is discussed in Section 3.1. Tasks performed in this phase are discussed in Section 4.

The third phase of the FDBS evolution, *federated database system operations*, involves managing and manipulating multiple integrated databases using an FDBMS. The processors support various run-time FDBS operations (e.g., query processing and transaction management). The processors and the federated schemas developed or generated in the second phase allow the selective, shared, and consistent access to data stored in the component DBSs. An FDBMS provides an interface to the users and applications and may allow execution of ad hoc queries. Tasks performed in this phase are discussed in Section 5.

3.1 Methodology for Developing a Federated Database System

The methodology discussed here extends the methodology of Sheth [1988b] for developing schemas for an FDBS to include processors. Developing a new FDBS primarily consists of integrating existing component databases. A *bottom-up* FDBS development process can be followed for this purpose. This process can also be used for adding a new component database to an FDBS.

When new applications are developed using an existing FDBS, it is necessary to determine whether the data requirements

of the application are supported by a federated schema. If they are not, it is necessary either to extend a federated schema or create a new federated schema and either to extend an existing component database or create a new component database. This process is called a *top-down* FDBS development process. It is an extension of the traditional distributed database design process. In practice, elements of both the bottom-up and top-down processes are used to develop an FDBS.

A data dictionary/directory (DD/D) often plays an important role in coordinating various activities by storing essential information. In addition to storing all schemas representing information about the data managed by the FDBS, a DD/D also stores mappings among schemas, information about schemas and databases (such as statistics relevant to query optimization), schema-independent information (such as tables and functions for unit/format conversions or heuristics for query optimization), and various types of system informations (such as capabilities of each component DBMS, network addresses of each system hosting a component DBMS, and communication facility to be used to communicate with a given system).

3.1.1 Bottom-Up Development Process

A bottom-up FDBS development process is used to integrate several existing databases to develop an FDBS. Figure 21 illustrates the bottom-up process outlined below:

(1) **Translate schemas:** Translate the local schema of a component database into a component schema expressed in the CDM. Generate the mappings between the objects in the two schemas. Develop (or identify if one already exists) the transforming processor that can transform commands expressed on the component schema into the commands expressed on the corresponding local schema.

(2) **Define export schemas:** Define export schemas from a component schema. This step is performed by the administrators (component DBAs) of respective component DBSs to authorize part of their databases to be included in the FDBS based on the negotiations with the federation DBA. Develop (or identify if one already exists) the appropriate filtering processor.

(3) **Integrate schemas:** Select a related set of export schemas to be integrated and integrate them. Integration of each set of export schemas will produce one federated schema. Develop (or identify if one already exists) a constructing processor that would transform the commands expressed on federated schemas into commands expressed on the corresponding export schemas. This includes generating mappings with appropriate distribution information. This step is repeated once for each related set of export schemas and the corresponding federated schema.

(4) **Define external schemas:** If necessary, define external schemas for each federation user or class of federation users. Build or identify the necessary filtering and transforming processors. The transforming processor performs schema translation if the data model of the external schema is different than the CDM.

3.1.2 Top-Down Development Process

A top-down FDBS development process is used when an FDBS already exists and additional user requirements (e.g., to support a new application) are placed on it. Figure 22 illustrates the top-down process outlined below:

(1) **Define or modify external schemas:** Collect federation user requirements and analyze them to define new external schemas or extensions to the existing external schemas.

(2) **Analyze schemas:** Compare relevant federated schemas with the external schemas to identify parts of the external schemas that are already in the federated schema and hence supported by the FDBS. If some part of an external schema is not already supported by

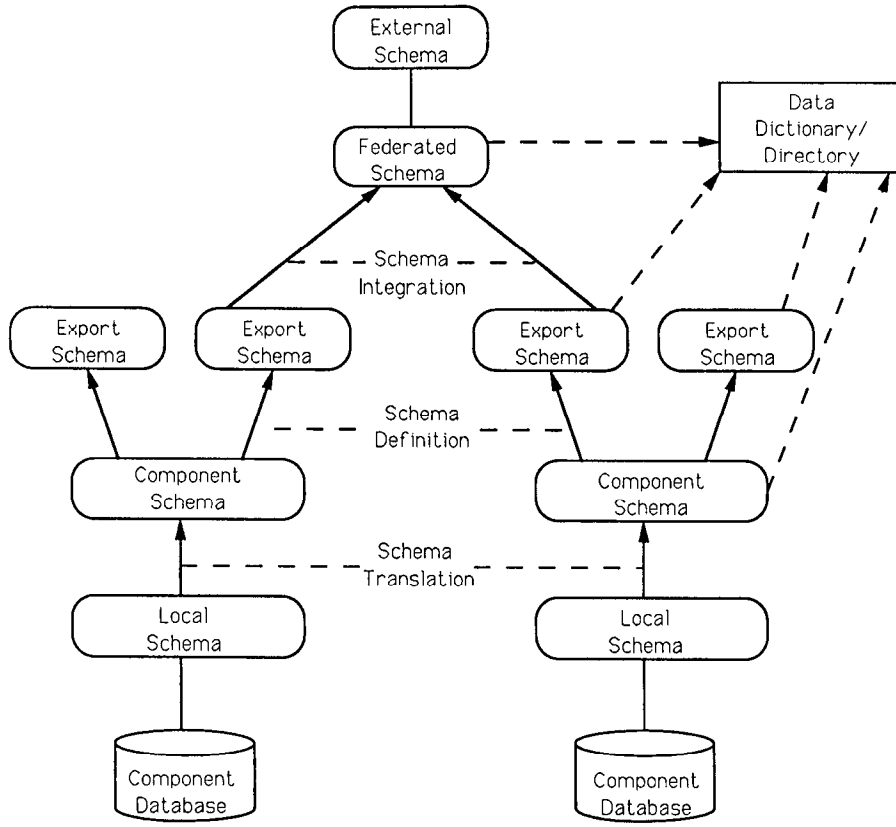

Figure 21. Bottom-up FDBS developing process.

the FDBS, a federated schema will need to be extended or developed to include that part. We refer to this unsupported part as a temporary schema (which is discarded at the end of the integration process). One or more of the component databases will have to support the temporary schema. This can be accomplished in one of three ways:

(a) The required data exist in one or more component database(s). In this case, identify the component schemas containing the description of required data and negotiate with their administrators to have a description of this data placed in an export schema with appropriate access rights.

(b) The required data are not implemented in any component database, and a component DBA is willing to place the required data in his or her component database. In this case, local, component, and export schemas of the relevant databases are modified.

(c) The required data are not implemented in any component database, and no component DBA is willing to place the required data in his or her component databases. In this case, the temporary schema is implemented as a separate database of an existing component DBMS. Alternatively, a new component DBMS may be used that may require a new transforming

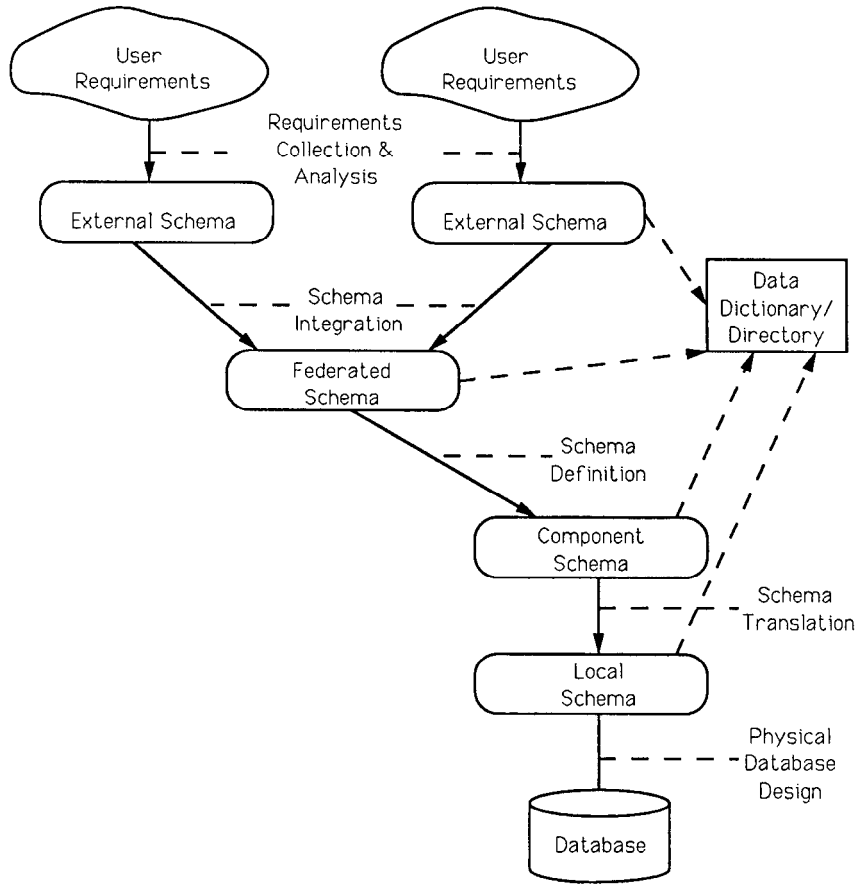

Figure 22. Top-down FDBS developing process.

processor. The temporary schema becomes a new component schema.

(3) **Integrate schemas:** Integrate the temporary schema with the relevant federated schema and discard the temporary schema.

4. FEDERATED DATABASE SYSTEM DEVELOPMENT TASKS

Many tasks involved in developing a centralized or a distributed DBS [Ozsu and Valduriez 1990, Chapter 5; Teorey 1990] are also relevant to developing an FDBS and can be adapted with minor changes. In this section, we discuss four tasks that do not typically arise in that context but have particular significance for developing an FDBS. They are schema translation, access control, negotiation, and schema integration. As identified in Section 3.1, these tasks are relevant to the development of the schemas and are affected by the data requirements of the federation users.

4.1 Schema Translation

Schema translation is performed when a schema represented in one data model (the source schema) is mapped to an equivalent schema represented in a different data model (the target schema). Schema translation is needed in two situations:

- Translating a local schema into a component schema when the DBMS's native data model is different from the CDM.
- Translating a part of the federated schema into an external schema when

the external schema is expressed in a data model different than the CDM.

For example, assume that if the CDM is an extended Entity Relationship (EER) model, then one of the component DBMSs to be integrated is a relational DBMS and another is a CODASYL DBMS, and a class of federated users wants a relational view. The following are required: (1) translation of the local schema of the relational DBMS into its equivalent component schema expressed in the EER model, (2) translation of the local schema of the CODASYL DBMS into an equivalent component schema expressed in the EER model, and (3) translation of a part of the federated schema in the EER model into an external schema expressed in the relational model. Two general approaches for performing schema translation are discussed below.

The first approach develops explicit mappings between each source and the target schema. This approach is appropriate when a (class of) federation user(s) requires specific data structures in the target schema. The DBA must then specify mappings that transform the source schema to the target schema. There are three cases:

- All schema objects in the target schema can be derived from the schema objects in the source schema, and there exist inverses for all of these mappings. The DBA specifies the mappings and their inverses.
- All schema objects in the target schema can be derived from the schema objects in the source schema, but inverses may not exist for the mappings. If the update were to be allowed, the DBA must construct a filtering processor to solve the associated view update task.
- If the target schema contains objects that cannot be derived from the objects in the source schema, the DBA must modify the target schema or construct a constructing processor that integrates schema objects from other schemas to support all of the objects in the target schema.

The second approach develops mapping rules to generate the target schema from the source schema. These rules specify how each object in the target schema is derived from objects in the source schema. If the mapping rules do not have inverses, the view update problem must be solved. The following is a simplified example of a set of mapping rules that can be used to generate a relational schema from a CODASYL schema:

(1) Each record type is mapped to a table with the same name.
(2) Each field in a record type *A* is mapped to a column of table *A*.
(3) Each record identifier in record type *A* is mapped to a key in table *A*.
(4) Each set (one-to-many relationships between record type *A* and record type *B*) is mapped to a column in table *B* that contains values from the key of table *A*. This column is called a *foreign key*.

Consider the CODASYL schema in Figure 23(a) (from Larson [1983a]). The COMPANY record type has two fields, COMPANY-NAME and CITY. The PRODUCT record type has two fields, PRODUCT-NAME and COST. COMPANY-NAME is the unique record identifier for the COMPANY record type, and PRODUCT-NAME is the unique record identifier for the PRODUCT record type. There is a PRODUCES set consisting of the COMPANY record type as the owner and the PRODUCT record type as the member. The PRODUCES set maintains a one-to-many relationship between COMPANY and PRODUCT.

The relational schema of Figure 23(b) results when the above mapping rules are applied. The COMPANY table has two fields, COMPANY-NAME and CITY. The PRODUCT table has three fields, PRODUCT-NAME, COST, and COMPANY-NAME. COMPANY-NAME is the key for the COMPANY table, and PRODUCT-NAME is the key for the PRODUCT table. The COMPANY-NAME column of PRODUCT table is a foreign key; it contains only values from the COMPANY-NAME key field of the COMPANY table.

Zaniolo [1979] developed a tool that automatically generates relational schemas

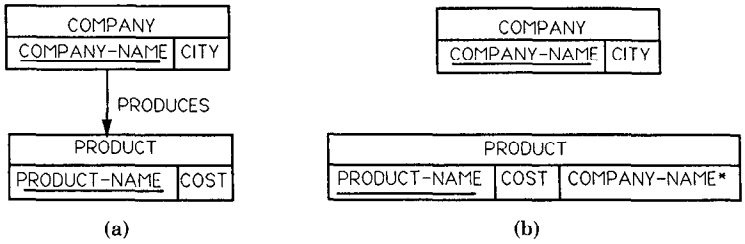

Figure 23. Equivalent CODASYL and relational schemas. (a) CODASYL schema; (b) relational schema. * —Foreign key.

from CODASYL schemas. Elmasri et al. [1985] and Teorey et al. [1986] discuss transformations between extensions of the Entity Relation data model [Chen 1976] and the relational data model. Lien [1981] describes mappings from the hierarchical to the relational data model. Tsichritzis and Lochovsky [1982] provide a summary of these types of mappings.

In practice, the translation task may require more than just data model translation because the source and the target schemas may not be able to represent exactly the same semantics. Hence schema translation poses two contradictory requirements: (1) Capture additional semantics during the schema translation that can later help in the tasks of schema integration and view update, and (2) maintain only the existing semantics because the local schema is not able to support the additional semantics. These requirements are discussed next.

Using a semantic data model as the CDM can facilitate representation of additional semantics that may be difficult or impossible to specify in a traditional model such as the relational model. As an example, a generalization relationship with inheritance between two entity types can be explicitly represented in a component schema that uses a semantic data model. As another example, a foreign key between relations *R1* and *R2* in a local schema expressed in the relational model may be explicitly represented as a relationship between the two entities representing *R1* and *R2* in the corresponding component schema in an EER model. One, however, should be careful about the additional semantic information provided during the translation from a local schema to a component schema. This is because a component DBMS is autonomous, and the database managed by it should not be changed as a result of the translation process. In other words, the translation should be reversible in the sense that (a) the component schema in the CDM should represent the same database represented by the local schema and (b) it should be possible to translate a command on a component schema into command(s) on the corresponding local schema. A notion that may be useful in this context is that of *content-preserving transformations* [Rosenthal and Reiner 1987].

4.2 Access Control

An FDBS should be designed to control access to component databases by federation users. The system architecture of an FDBS (shown in Figure 11) has filtering processors at two levels. Each can be used to provide access control. The filtering processors relating the export and the component schemas control access to component DBSs. The filtering processor relating the external and federated schemas controls access to the federated schemas.[7] Negotiation between the component and federation DBAs may be necessary to reach an agreement on how to control the data a component DBA wants to keep secure from some of the federation users while allowing access to other federation users. Alternately, a new federated schema can be established

[7] This is similar to using the view mechanism for access control security in centralized and distributed DBMSs [Bertino and Haas 1988].

for use by a selected subgroup of the original federation. For example, suppose component databases each contain information about commercial shipping and military shipping. The following approaches are possible:

- Each component DBA includes schema objects describing both commercial and military shipping in their respective export schemas. This information is integrated into a single federated schema. The federation DBA creates two external schemas, one for accessing commercial shipping information and one for accessing military shipping. In this scenario, the local DBAs trust the federation DBA to provide appropriate access controls on each external schema.
- Each component DBA generates two export schemas, one containing the schema objects describing commercial shipping and the other containing schema objects describing military shipping. Two federated schemas are created, one dealing with commercial shipping and the other dealing with military shipping. In this scenario, the component DBAs control appropriate access on each export schema and hence is preferred to the previous one.

Other access control and security issues include the following:

- How users are identified, grouped into classes, and named for access control security purposes. Templeton et al. [1987a] discuss the issue of installing a new user.
- How data are identified, grouped into classes, and named for access control security purposes [Abbott and McCarthy 1988; Templeton et al. 1987b].
- What operations are used for controlling security privileges [Fagin 1978; Larson 1983b; Selinger and Wade 1976].

A case study of the access control features of Mermaid [Templeton et al. 1987a] is interesting. It uses access control at four levels. First, a user must have an account on the computer where the Mermaid user interface can be run. Second, the user should be registered with Mermaid. Mermaid maintains an *access file* on each computer for this purpose. Access files control access to Mermaid's data dictionary and directory. Third, Mermaid maintains an *access list* that identifies the external and federated schemas each user is authorized to use. Fourth, the user must be registered with each computer having a component DBS that is relevant to the user's federated schema. In addition, Mermaid provides extensive run-time support for access control, uses data encryption (for passing access control information over the network and for storing it in access files and access lists), and provides an audit trail and a journal trail for security purposes.

One of the heterogeneities that may be encountered in an FDBS is the existence of different and incompatible mechanisms for expressing and enforcing access control policies. An access control mechanism used by an FDBMS, such as the view mechanism, may also conflict with the autonomy of the component DBMSs. A solution that uses preprocessing is discussed by Wang and Spooner [1987].

4.3 Negotiation

A federation DBA manages federated schemas. A component DBA manages the export schemas defined over the component DBS he or she manages. A federation DBA and component DBAs must reach an agreement about the contents of the export schemas and operations allowed on the export schemas such that federated schemas can be defined over them to support federation users. The dialogue between the two administrators to reach this agreement is called *negotiation*. To facilitate negotiation, the administrators follow protocols governing the messages exchanged during a negotiation.

In terms of the reference schema architecture, there are two ways to perform negotiation. First, a component DBA may allow the federation DBA to read the component schemas he or she controls but does not give any specific rights for data access. When a federation DBA determines data access requirements, he or she sends a request to the component DBA to define an

export schema with appropriate access rights. This request typically includes information on the data to be accessed, the types of access (retrieval or update), and the federation users who will access the data. In some cases, it may also include requirements on the component DBS such as constraints to be satisfied (e.g., how current or consistent the data must be and maximum response time), and whether the federation DBA or user may grant others some or all of the access rights for this export schema [Larson 1983b]. The component DBA may accept or refuse any part of the request for any reason in defining the export schemas. In the second alternative, all component DBAs may define the export schemas a priori and require the access from the FDBMS to be limited to the contents and access rights contained in those export schemas.

Two of the many aspects of FDBS operations for which negotiation protocols need to be defined are the following:

- A component DBA decides to withdraw access to a schema object or change a schema object in a local schema.
- A federation user decides that access to a schema object is no longer needed.

Heimbigner and McLeod [1985] present negotiation protocols for a multisited distributed dialogue among the DBAs in an FDBS. Alonso and Barbara [1989] consider the case in which the federated schema is a materialized view (they call it *quasi-copy*). In this context, they explore ways to express the needs of a federation DBA precisely, to determine the degree of sharing the component DBA is willing to offer, and to estimate the cost of a specific agreement for both the component DBS and the FDBS.

4.4 Schema Integration

View integration refers to integrating multiple user views into a single schema (e.g., federated schema development in a top-down FDBS development process). *Schema integration* refers to integrating (usually existing) schemas into a single schema (e.g., federated schema development by integrating export schemas in a bottom-up FDBS development process). The tasks are quite similar and are treated uniformly as schema integration in this paper.

Many approaches and techniques for schema integration have been reported in the literature. The survey paper of Batini et al. [1986] discusses and compares 12 methodologies for schema integration. It divides schema integration activities into five steps: preintegration, comparison, conformation, merging, and restructuring. In the context of FDBS development, preintegration activities involve translation of schemas into a CDM so they can be compared and specification of global constraints and naming conventions. The latter involves activities that may be useful in the comparison step (e.g., specifying a thesaurus that may be used for identifying naming conflicts, homonyms, or synonyms). Although schema translation has been studied independently from schema integration, the two tasks are highly interrelated. Schema translation may be more constrained when integrating existing DBSs than during view integration because the constraints of the existing data structures cannot be changed.

The comparison step (also called schema analysis) involves two activities: (a) analyzing and comparing the objects of the schemas (and possibly databases) to be integrated, including identification of naming conflicts (e.g., homonym and synonym detection), domain (i.e., value type) conflicts, structural differences, constraint differences, and missing data and (b) specifying the interrelationships among the schema objects. The conforming step is closely tied to the comparing step since it is difficult to compare unless the related information is represented in a similar form in different schemas.

Unless the schemas are represented in the same model, analyzing and comparing their schema objects is extremely difficult. It is important to note that comparison of the schema objects is primarily guided by their semantics, not by their syntax. Hence the choice of the CDM is critical. The CDM should be semantically rich; that is, it should provide abstraction and constraints

so that the semantics relevant to schema integration can be presented. Thus, semantic (or conceptual) data models are much preferred over traditional data models such as relational, network, or hierarchical. Dayal and Hwang [1984] suggest that the concept of generalization is important for schema integration. One reason is that similar or related concepts are represented at different levels of abstractions in different schemas. For example, one schema may contain attributes Office_Phone_Number and Home_Phone_Number, both of which can be shown as specializations of the attribute Telephone_Number in another schema.

Analyzing and comparing schema objects is followed by specifying the interrelationships among the schema objects. Consider integrating two EER schemas. In this case, there are three types of schema objects: entity types, relationship types, and attributes. In the methodology discussed by Elmasri et al. [1986] as well as in several other schema integration methodologies, relationships among attributes in the two schemas are specified first. Relationships among other object types (e.g., the entity types and the relationship types) follow.

Two attributes, *a1* and *a2*, may be related in one of the three ways [Larson et al. 1989; Sheth and Gala 1989]: *a1 is_equivalent_to a2*, *a1 includes* (or *is_included_in*) *a2*, or *a1 is_disjoint_with a2*. Determining such relationships can be time consuming and tedious. If each schema has 100 entity types, and an average of five attributes per entity type, then 250,000 pairs of attributes must be considered (for each attribute in one schema, a potential relationship with each attribute in other schemas should be considered). Sheth and Gala [1989] argue that this task cannot be automated, and hence we may need to depend on heuristics to identify a small number of attribute pairs that may be potentially related by a relationship other than *is_disjoint_with*.

Two entity types, *E1* and *E2*, may be related in one of five ways [Elmasri et al. 1986; Navathe et al. 1986]: *E1 equals E2*, *E1 includes* (or *is_included_in*) *E2*, *E1 overlaps* (or *may_be_integratable_with*) *E2*, *E1 is_disjoint_but_integrable_with E2*, and *E1 is_disjoint_and_nonintegrable_with E2*. Similar relationships exist among relationship types. Elmasri et al. [1986] and Sheth et al. [1988b] use the relationships among attributes in a heuristic algorithm to identify pairs of entity types and relationship types that may be related by the first three types of relationships. The actual relationships are specified by the person (the DBA in a tightly coupled FDBS, a user in a loosely coupled FDBS) integrating the schemas. Using *classification* in the CANDIDE semantic data model [Beck et al. 1989] and the relationships among attributes, Sheth and Gala [1989] suggest that many of the relationships among entity types and relationship types can be automatically discovered without additional human input.

In [Sheth et al. 1988b], specification of relationships among schema objects is used as an input to a lattice generation or merging activity to generate a single integrated schema. Other efforts, particularly those that do not rely on identifying relationships among schema objects, as discussed above, provide a set of operators for the DBA to correlate or integrate the objects to generate an integrated schema [Motro and Buneman 1981].

One reason why a completely automatic schema integration process (particularly for discovering attribute relationships) is not possible is because it would require that all of the semantics of the schema be completely specified. This is not possible because, among other reasons, (1) the current semantic (or other) data models are unable to capture a real-world state completely, (2) it will be necessary to capture much more information than is typically captured in a schema, and (3) there can be multiple views and interpretations of a real-world state; and the interpretations change with time. Convent [1986] formally argues that integrating relational schemas is undecidable.

Three tools developed to perform schema integration are reported in Hayes and Ram [1990], Sheth et al. [1988b], and Souza [1986]. Sheth et al. [1988b], for example, describe a forms-based interactive tool to integrate EER schemas. It accepts the

definitions of the schemas to be integrated, guides the integrator through the process of defining attribute equivalencies, uses attribute equivalencies to rank entity type and relationship type pairs that may be related heuristically, accepts assertions about the relationships among the entity types and relationship type pairs, checks for their consistency, and performs the merging task automatically.

After schemas have been integrated, it may be necessary to decide how to allocate the data among multiple component DBSs. This distribution design task has been studied extensively [Ceri et al. 1987].

5. FEDERATED DATABASE SYSTEM OPERATION

The previous section discussed some of the important tasks for developing an FDBS. Many tasks relevant to the operation (i.e., run-time system) of distributed DBMSs are also relevant to the operation of multidatabase systems and FDBSs. In this section, we discuss four tasks that are either specific to an FDBS (or a multi-DBMS system) or are significantly different from similar tasks in a distributed DBMS. These tasks are application independent.

5.1 Query Formulation

The same query languages used in centralized and distributed DBMSs can be used for formulating queries in a tightly coupled FDBS. This is because a lightly coupled FDBS provides location, distribution, and replication transparencies. Most loosely coupled FDBSs provide a multidatabase language to allow a federation user to access data from multiple component DBSs. A multidatabase language provides functions that are not present in data manipulation languages used in centralized and distributed DBMSs [Litwin et al. 1987]. It provides a capability to define federated schemas as views over multiple export schemas (or parts of component schemas) and to formulate queries against such a view. In addition the language deals with problems identified in the discussion on schema integration, such as naming conflicts and data structure/type/scale differences. Some systems using multidatabase languages can define multiple semantics over the same data (e.g., dynamic attributes in Litwin and Abdellatif [1986]). Examples of multidatabase languages include MDSL [Litwin and Abdellatif 1987], MSQL [Litwin et al. 1987], GSQL [Jacobs 1985], a relational language with extended abstract data types [Czejdo et al. 1987], and a graphical multidatabase query language [Rusinkiewicz et al. 1989]. CALIDA [Jacobsen et al. 1988]) provides a menu-based interface to formulate queries against multiple component DBMSs.

5.2 Command Transformation

A command transformation processor translates commands in one language, called the source language (or operations in one data model, called the source data model) into commands in another language, called the target language (or operations in another data model, called the target data model). Converting between procedural and nonprocedural languages is of particular interest.

5.2.1 Converting a Nonprocedural Language into a Procedural Language

A common example of this type of conversion is transforming relational algebraic operations into a sequence of CODASYL programming language statements. Consider the relational operation *join* that causes two tables to be combined. Two rows, one from each table, are joined if their values satisfy a specified Boolean (the join condition) condition. Using the example of Section 4.1, joining the COMPANY and PRODUCT tables where the COMPANY-NAME of a COMPANY equals the COMPANY-NAME of a PRODUCT results in a single table consisting of columns of both the COMPANY and PRODUCT tables. Using code generation techniques, it is possible to automatically generate code consisting of two nested loops. The outer loop reads successive COMPANY records. The inner loop reads successive PRODUCT records in the PRODUCES set (owned by COMPANY) and constructs rows of the joined table.

This idea can be generalized as follows. For each relational operation, *R*, develop an algorithm, *A*, which generates equivalent CODASYL code. For each nested operation, *R1* and *R2*, develop an algorithm for using the corresponding generation algorithms *A1* and *A2* to generate an equivalent CODASYL code. For example, if a query involves joining the DISTRIBUTOR table with the result of joining the COMPANY and PRODUCT tables, then generate two nested loops that perform the join of COMPANY and PRODUCT and place those two loops inside another loop that generates the join of DISTRIBUTOR table with the result of joining COMPANY and PRODUCT.

Another approach involves generating all possible strategies for performing nested relational operations, then selecting the best strategy. Heuristics can be used to generate only promising strategies, decreasing the time to perform optimization at the cost of producing a near-optimal rather than optimal strategy. Rules for converting one strategy into a better strategy are described by Chu and Hurley [1982] and Ceri and Pelagatti [1984].

An approach taken by some researchers is to use an intermediate language that can serve as an umbrella for multiple target languages and be able to express most, if not all, of the operations expressed in the target languages. Piatetsky-Shapiro and Jakobson [1987] define a language called DELPHI that combines the power of relational algebra with many additional database operations, including grouping, sorting, aggregates, and nested queries. They also describe rule-based transformations to convert DELPHI to different target query languages such as SQL and fourth-generation languages. In CALIDA [Jacobson et al. 1988], which is a loosely coupled FDBS, a user's interactions with a menu-based interface results in a query in DELPHI, which is then converted into queries in the desired target language.

5.2.2 *Converting a Procedural Language into a Nonprocedural Language*

An example of converting a procedural language into a nonprocedural language is synthesizing relational algebraic operations from a sequence of CODASYL programming language statements. This problem is quite difficult. Demo [1983] suggests a data flow analysis approach that may be used for some cases. Solving the general case is an open research problem. Fortunately, the need for this type of transformation seldom arises because a procedural language is seldom chosen for the federation level.

5.3 Query Processing and Optimization

In a loosely coupled FDBS, the FDBMS can support little or no query optimization (see Section 2.1). In a tightly coupled FDBS, the FDBMS can perform extensive query optimization. Query processing involves converting a query against a federated schema into several queries against the export schemas (and the corresponding component DBSs) and executing these queries. Query processing in an FDBMS is similar to that in a distributed DBMS (see e.g., [Yu and Chang 1984] for a survey of query-processing techniques for distributed DBMSs). In an FDBMS, however, the following additional complexities may be introduced due to heterogeneity and autonomy:

- The cost of performing an operation may be different in different component DBSs. Due to autonomy of a component DBS, the cost of performing the operation in a component DBMS may not be known or may only be approximately known. In addition, this cost may vary from time to time as the local system load changes.
- The component DBMSs may differ in their abilities to perform local query optimizations.
- The system and database operations provided by each of the component DBMSs and the FDBMSs may be different. Examples of system operations include the ability to create temporary relations and the ability to receive and reference data from other sites. Examples of database operations include the different relational algebraic operations (join, selection, etc.) and aggregations. Not all component DBMSs may support the

same operations and aggregations (or their semantics may vary, e.g., one component DBMS may remove duplicates but another may not).

Landers and Rosenberg [1982] discuss optimization problems and solutions adopted for some of the above issues in Multibase. Mermaid has implemented and tested comprehensive algorithms for query optimization [Chen et al. 1989] that involve a dynamically changing network environment and different processing costs at different component DBMSs.

A query evaluation plan can be formulated as a program in an intermediate task specification language and then executed. DOL (Distributed Operation Language) [Rusinkiewicz et al. 1990] is an example of such a language. Its main functions include invocation of local and remote tasks (processes), synchronization of their execution, data exchange between tasks (including reformatting), and exception handling. The commands for the local systems can be embedded in the DOL programs that are forwarded to the local systems and executed under local control.

Query processing in an FDBMS can be divided into *global* processing performed by the Global Data Manager (GDM) (a constructing processor) of an FDBMS and *local* processing performed by the Local Database Interface (LDI) (a transforming processor) associate with a component DBS. Global query processing and optimization relate to processing a query or transaction submitted by a federation user, called a *global transaction*, and dividing it into multiple subtransactions, called *local transactions*, for the LDIs of the appropriate component DBSs. Local query processing and optimization relate to processing a local transaction at a single component DBS. Each is discussed below.

Global optimization involves evaluating the following trade-offs:

- The amount of work done by the GDM and the complexity of the LDI, and
- The amount of communication and processing done by different component DBSs.

Similar trade-offs exist in a distributed DBMS, but the heterogeneity and autonomy add considerably to the complexity. Three FDBMS query optimization design approaches are as follows:

- **Simple LDIs and GDM:** The GDM transforms the global transaction into the smallest possible local transactions. An LDI (and hence a component DBS) receives multiple local transactions for each global transaction. The LDI sends the result of each local transaction to the GDM. The GDM merges all the results. Multibase takes this approach. It results in heavy workload for the GDM and significant communication between the GDM and the LDIs.
- **Medium complexity of the GDM and LDIs:** The GDM transforms the global transaction into a set of the largest possible local transactions (one for each relevant component DBS). This reduces the GDM workload and communication between the GDM and the LDIs. Fewer local transactions are sent to LDIs, and the data returned to the GDM is a result of processing a more complete local transaction).
- **Complex GDM and LDIs:** In this case, GDM generates efficient programs that involve participation of LDIs in the global optimization. Partial results can be sent to the GDM or other LDIs. LDIs have to support additional functionalities like sorting, removing duplicates, and handling and merging temporary files. DDTS and Mermaid take this approach. It results in a further reduction in communication and GDM workload. Rusinkiewicz and Czejdo [1987] discuss an algorithm that attempts to maximize local processing and minimize data transfer between component DBSs.

Local optimization involves optimizing execution of local transaction received from the GDM. An FDBMS provides additional requirements as well as opportunities for local optimization as follows:

- Query languages for the component DBMSs may be different. Thus, operation transformation for different pairs of

the query language used by the FDBS and the query language provided by a component DBMS may involve different techniques.

- Physical database organization of each of the component DBSs may be different. Similarly, criteria for access path selection for different component DBMSs may be different. Due to the autonomy of the component DBMS, however, the GDM or an LDI may not have enough information on these matters.

Onuegbe et al. [1983] and Dayal and Goodman [1982] develop strategies for local query optimization in DDTS and Multibase, respectively.

5.4 Global Transaction Management

The Global Transaction Manager (GTM) is responsible for maintaining database consistency while allowing concurrent updates across multiple databases. Supporting global transaction management in an environment with multiple heterogeneous and autonomous component DBSs is very difficult. This is underlined by the fact that none of the prototype FDBMSs support global transaction management. Nowhere does the autonomy of component DBMSs present more problems than in supporting updates.

There are two types of transactions to be managed: global transactions submitted to the FDBMS by federation users and local transactions directly submitted to a component DBMS by local users. The basic problem in supporting global concurrency control is that the FDBMS does not know about local transactions since a component DBMS is autonomous. That is, local wait-for relationships are known only to the transaction manager of the component DBMS. Without knowledge about local as well as global transactions, it is highly unlikely that efficient global concurrency control can be provided. Due to the existence of local transactions, it is very difficult to recognize when the *execution order* differs from the *serialization order* at any site [Du et al. 1989]. Additional complications occur when different component DBMSs and the FDBMS support different concurrency control mechanisms [Gligor and Popescu-Zeletin 1986]. The problem of global deadlock detection must also be addressed [Logar and Sheth 1986].

Several researchers are currently studying these problems [Alonso et al. 1987; Breitbart and Silberschatz 1988; Elmagarmid and Helal 1988; Pu 1987]. As argued by Barker and Ozsu [1988] and Du et al. [1989], however, the published solutions often make unrealistic and pessimistic assumptions, support a low level of concurrency, and/or sacrifice autonomy in order to obtain higher concurrency.

It is unlikely that a theoretically elegant solution that provides conflict serializability without sacrificing performance (i.e., concurrency and/or response time) and availability exists. One often accepted trade-off is to limit the functionality of concurrency control in favor of preserving site autonomy. Examples of this would be to allow only unsynchronized retrievals, preclude multisite updates, or perform local updates off-line. Another approach is to devise mechanisms that specifically suit the limitations of a given environment and provide the required level of consistency. Eliassen and Veijalainen [1987] propose a concept of *S-Transactions* (for semantic transactions) suited for a banking environment consisting of a network of highly autonomous systems. It may be desirable to devise solutions that do not meet the conflict serializability criteria but that are practical and meet a desired level of consistency. Du and Elmagarmid [1989] propose a weaker consistency criterion called *Quasi-Serializability*[8] that works provided there are no value dependencies (e.g., referential integrity constraints) across databases. Garcia-Molina and Salem [1987] and Alonso et al. [1987] propose a concept of *Sagas* that provides semantic atomicity but does not serialize execution of global transactions. More work on weaker consistency

[8] A *Quasi-Serializable schedule* is one in which the local transaction schedules are serializable and the global transactions are executed serially. We need to understand the practical significance of this and other proposed weaker consistency criteria better.

criteria and a better understanding of their significance in practical terms is needed. Techniques to specify and execute the transactions that selectively provide atomicity, isolation, and durability properties need to be further researched. Little attention has been given to the problems of fault tolerance, ensuring the integrity of redundant data, and supporting integrity constraints across component databases.

6. FUTURE RESEARCH AND UNSOLVED PROBLEMS

We discussed the concept of federation in the context of database systems. A federated database system is a collection of cooperating but autonomous and possibly heterogeneous database systems. A reference architecture was used to study various FDBS architectural alternatives and their implications. A methodology for developing FDBSs, particularly the tightly coupled FDBSs, was discussed. Finally, we discussed important tasks that need to be performed in order to develop and operate FDBSs.

Several problems need further research and development. They include the following:

- Identifying and representing all semantics useful in performing various FDBS tasks such as schema translation and schema integration and determining contents of schemas at various levels.
- Lack of software tools to aid in performing various FDBS tasks with a high degree of automation and an integrated toolset for developing, maintaining, and managing FDBSs.
- Lack of adequate transaction management algorithms that provide a specified level of consistency (i.e., are correct with respect to a given consistency criteria) and fault tolerance at acceptable performance within the heterogeneity and autonomy constraints of an FDBS.
- How to address management and efficiency issues related to autonomy of component DBSs.

The focus of the past activities in FDBSs has been on databases that store more structured data, often called business data. Recently, however, there has been a great deal of activity for constructing DBMSs that manage data for the so-called nontraditional applications. These applications require less structured data in the forms such as text, image, graphics, and voice. Current database research activities, particularly related to object-oriented database systems, address centralized management of these types of data. We now need to investigate issues in integrating such systems [Sheth 1987b, 1988b].

The second significant extension is to create information systems that not only include database systems but also application programs and expert systems. Such a system may be called a federated knowledge base system. One of the main problems is to represent information contents, processing capabilities, and semantics (including behavioral aspects) of programs and expert systems adequately. We need to define a description that plays a role with respect to an application program that is equivalent to the role a schema plays for a database. One such effort is that of a "capability schema" for an application program [Ryan and Larson 1986]. Models for federation also need to be developed for such environments.

ACKNOWLEDGMENTS

We thank the referees, S. March, M. Rusinkiewicz, G. Thomas, and M. Templeton, who have helped us toward the current version; their time and care is gratefully acknowledged.

REFERENCES

ABBOTT, K., AND MCCARTHY, D. 1988. Administration and autonomy in a replication-transparent distributed DBMS. In *Proceedings of the 14th International Conference on Very Large Data Bases* (Aug.), pp. 195–205.

ALONSO, R., AND BARBARA, D. 1989. Negotiating data access in federated database systems. In *Proceedings of the 5th International Conference on Data Engineering* (Feb.), pp. 56–65.

ALONSO, R., GARCIA-MOLINA, H., AND SALEM, K. 1987. Concurrency control and recovery for global procedures in federated database systems. In *Q. Bull. IEEE-CS TC Data Eng. 10*, 3 (Sept.), 5–11.

BARKER, K., AND OZSU, T. 1988. A survey of issues in distributed heterogeneous database systems.

Tech. Rep. TR 88-9, Univ. of Alberta Edmonton, Canada.

BATINI, C., LENZERINI, M., AND NAVATHE, S. 1986. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv. 18*, 4 (Dec.), 323–364.

BECK, H., GALA, S., AND NAVATHE, S. 1989. Classification as a query processing technique in CANDIDE semantic data model. In *Proceedings of the 5th International Conference on Data Engineering* (Feb.), pp. 572–581.

BELCASTRO, V., ET AL. 1988. An overview of the distributed query system DQS. In *Proceedings of the International Conference on Extending Data Base Technology* (Venice, Italy, Mar.). In *Computer Science*. Vol. 303, Springer-Verlag, New York, pp. 170–189.

BERTINO, E., AND HAAS, L. 1988. Views and security in distributed database management systems. In *Proceedings of the International Conference on Extending Database Technology* (Venice, Italy, Mar.). In *Computer Science*. Vol. 303, Springer-Verlag, New York, pp. 155–169.

BLAKEY, M. 1987. Basis of a partially informed distributed database. In *Proceedings of the 13th International Conference on Very Large Data Bases* (Brighton, UK, Sept.), pp. 381–388.

BREITBART, Y., AND SILBERSCHATZ, A. 1988. Multidatabase update issues. In *Proceedings of the ACM SIGMOD Conference* (June), 135–142.

BRZEZINSKI, Z., GETTA, J., RYBNIK, J., AND STEPNIEWSKI, W. 1984. UNIBASE–An integrated access to databases. In *Proceedings of the 10th International Conference on Very Large Data Bases* (Singapore, Aug.), pp. 388–395.

CARDENAS, A. 1987. Heterogeneous distributed database management: The HD-DBMS. In *Proc. IEEE 75*, 5 (May), 588–600.

CERCONE, N., MORGESTERN, M., SHETH, A., AND LITWIN, W. 1990. Resolving semantic heterogeneity. Panel at the *International Conference on Data Engineering* (Feb.).

CERI, S., AND PELAGATTI, G. 1984. *Distributed Databases—Principles and Systems*. McGraw-Hill, New York.

CERI, S., PERNICI, B., AND WIEDERHOLD, G. 1987. Distributed database design methodologies. In *Proc. IEEE 75*, 5 (May), 533–546.

CHEN, P. 1976. The entity-relationship model: Toward a unified view of data. *ACM Trans. Database Syst. 1*, 1 (Mar.), 9–36.

CHEN, A., BRILL, D., TEMPLETON, M., AND YU, C. 1989. Distributed query processing in Mermaid: A frontend system for multiple databases. *IEEE J. Selected Areas Commun. 7*, 3 (Apr.), 390–398.

CHU, W., AND HURLEY, P. 1982. Optimal query processing for distributed databases systems. *IEEE Trans. Comput. C-31* (Sept.), 835–850.

CONVENT, B. 1986. Unsolvable problems related to the view integration approach. In *Proceedings of the International Conference on Database Theory* (Rome, Italy, Sept.). In *Computer Science*, Vol. 243, Goos, G. and Hartmanis, J. Eds. Springer-Verlag, New York, pp. 141–156.

CZEJDO, B., RUSINKIEWICZ, M., AND EMBLEY, D. 1987. An approach to schema integration and query formulation in federated database systems. In *Proceedings of the 3rd International Conference on Data Engineering* (Feb.), pp. 477–484.

DATE, C. 1986. *An Introduction to Database Systems*, Vol. 1, 4th ed. Addison-Wesley, Reading, Mass.

DAYAL, U., AND GOODMAN, N. 1982. Query optimization for CODASYL database systems. In *Proceedings of the ACM SIGMOD Conference*, pp. 138–150.

DAYAL, U., AND HWANG, H. 1984. View definition and generalization for database integration in a multidatabase system. *IEEE Trans. Soft. Eng. SE-10*, 6 (Nov.), 628–644.

DE 1987. Special issue on federated database systems (mainly transaction management aspects). *Q. Bull. IEEE-CS TC Data Eng. 10*, 3 (Sept.).

DEMO, B. 1983. Program analysis for conversion from a navigational to a specification data base interface. In *Proceedings of the 9th International Conference on Very Large Data Bases* (Florence, Italy, Oct.), pp. 387–398.

DEVOR, C., ELMASRI, R., LARSON, J., RAHIMI, S., AND RICHARDSON, J. 1982b. Five-schema architecture extends DBMS to distributed applications. *Electron. Des.* (Mar. 18), 27–32.

DEVOR, C., ELMASRI, R., AND RAHIMI, S. 1982a. The design of DDTS: A testbed for reliable distributed database management. Tech. Rep. HP-82-273: 17–38, Honeywell Computer Sciences Center, Camden, Minn.

DP 1988. Special issue on heterogeneous distributed database systems. L. Lilien, Ed. *Distrib. Process. Tech. Comm. News. (Q.) 10*, 2 (Nov.).

DU, W., AND ELMAGARMID, A. 1989. Quasi serializability: A correctness criterion for global concurrency control in interbase. In *Proceedings of the 15th International Conference on Very Large Data Bases* (Amsterdam, Aug.), pp. 347–355.

DU, W., ELMAGARMID, A., AND KIM, W. 1990. Effects of local autonomy on heterogeneous distributed database systems. MCC Tech. Rep. ACT-OODS-EI-059-90, Microelectronics and Computer Technology Corp., Austin Tex.

DU, W., ELMAGARMID, A., LEU, Y., AND OSTERMANN, S. 1989. Effects of local autonomy on global concurrency control in heterogeneous database systems. In *Proceedings of the 2nd International Conference on Data and Knowledge Systems for Manufacturing and Engineering* (Oct.).

DWYER, P., AND LARSON, J. 1987. Some experiences with a distributed database testbed system. In *Proc. IEEE 75*, 5 (May), 633–647.

ELIASSEN, F. AND VEIJALAINEN, J. 1987. An S-transaction definition language and execution mechanism. Tech. Rep. No. 275, GMD, Hardenbergplatz, D-1000 Berlin 12, FRG.

ELIASSEN, F., AND VEIJALAINEN, J. 1988. A functional approach to information system interoper-

ability. In *Research into Networks and Distributed Applications (Proceedings of the EUTECO '88)*, Speth, R. Ed., Elsevier Science Publishers B.V., North-Holland, pp. 1121–1135.

ELLINGHAUS, D., HALLMANN, M., HOLTKAMP, B., AND KREPLIN, K. 1988. A multidatabase system for transaction autonomy. In *Proceedings of the International Conference on Extending Database Technology* (Venice, Italy, Mar.). In *Computer Science*, Vol. 303, Springer-Verlag, New York, pp. 600–605.

ELMAGARMID, A. 1987. When will we have true heterogeneous databases? (A position statement on Transaction Processing). In *Proceedings of the Fall Joint Computer Conference* (Dallas, Tex., Oct.), p. 746.

ELMAGARMID, A., AND HELAL, A. 1988. Supporting updates in heterogeneous distributed database systems. In *Proceedings on the International Conference on Data Engineering* (Feb.).

ELMASRI, R. 1981. GORDAS: A data definition, query and update language for the entity-category-relationship model of data. Tech. Rep. HR-81-250, Honeywell Inc., Camden, Minn.

ELMASRI, R., AND NAVATHE, S. 1989. *Fundamentals of Database Systems*. Benjamin/Cummings, Redwood City, Calif.

ELMASRI, R., LARSON, J., AND NAVATHE, S. 1986. Schema integration algorithms for federated databases and logical database design. Tech. Rep., Honeywell Corporate Systems Development Division, Camden, Minn.

ELMASRI, R., WEELDREYER, J., AND HEVNER, A. 1985. The category concept: An extension to entity-relationship model. *Data and Knowledge Engineering 1* (June). North-Holland, The Netherlands, pp. 75–116.

FAGIN, R. 1978. On an authorization mechanism. *ACM Trans. Database Syst. 3*, 3, 310–331.

GARCIA-MOLINA, H., AND KOGAN, B. 1988. Node autonomy in distributed systems. In *Proceedings of the International Symposium on Databases in Parallel and Distributed Systems* (Austin, Tex., Dec.), pp. 158–166.

GARCIA-MOLINA, H., AND SALEM, K. 1987. Sagas. In *Proceedings of the ACM SIGMOD Conference* (May), pp. 249–259.

GE, L., JOHANNSEN, W., LAMERSDORF, W., REINHARDT, R., AND SCHMIDT, J. 1987. Import and export of database objects in a distributed environment. In *Proceedings of the IFIP WG 10.3 Conference on Distributed Processing* (Amsterdam, Oct.).

GLIGOR, V., AND LUCKENBAUGH, G. 1984. Interconnecting heterogeneous database management systems. *Comput. 17*, 1 (Jan.), 33–43.

GLIGOR, V., AND POPESCU-ZELETIN, R. 1986. Transaction management in distributed heterogeneous database management systems. *Inf. Syst. 11*, 4, 287–297.

HAMMER, M., AND MCLEOD, D. 1979. On database management system architecture. Tech. Rep. MIT/LCS/TM-141, Massachusetts Institute of Technology, Cambridge, Mass.

HAMMER, K., AND TIMMERMAN, T. 1989. Automating data conversion between heterogeneous databases. Tech. Rep. ACA-ST-046-89, Microelectronics and Computer Technology Corp.

HAYES, S., AND RAM, S. 1990. Multi-user view integration system (MUVIS): An expert system for view integration. In *Proceedings of the 6th International Conference on Data Engineering* (Feb.).

HEIMBIGNER, D., AND MCLEOD, D. 1985. A federated architecture for information management. *ACM Trans. Off. Inf. Syst. 3*, 3 (July), 253–278.

HULL, R., AND KING, R. 1987. Semantic database modeling: Survey, applications, and research issues. *ACM Comput. Surv. 19*, 3 (Sept.), 201–260.

IEEE 1987. Special issue on distributed database systems. *Proc. IEEE 75*, 5 (May).

IISS 1986. The integrated information support system. *Gateway 2*, 2. Industrial Technology Institute (Mar.–Apr.).

JACOBS, B. 1985. *Applied Database Logic II: Heterogeneous Distributed Query Processing*. Prentice-Hall, Englewood Cliffs, N.J.

JACOBSON, G., PIATETSKY-SHAPIRO, G., LAFOND, C., RAJINIKANTH, M., HERNANDEZ, J. 1988. CALIDA: A knowledge-based system for integrating multiple heterogeneous databases. In *Proceedings of the 3rd International Conference on Data and Knowledge Bases* (Jerusalem, Israel, June), pp. 3–18.

KAUL, M., DROSTERN, K., AND NEUHOLD, E. 1990. ViewSystem: Integrating heterogeneous information bases by object-oriented views. In *Proceedings of the 6th International Conference on Data Engineering* (Los Angeles, Calif., Feb.), pp. 2–10.

KIM, W. 1989. Research directions for integrating heterogeneous databases. In *1989 Workshop on Heterogeneous Databases* (Chicago, Ill., Dec.).

LANDERS, T., AND ROSENBERG, R. 1982. An overview of Multibase. In *Distributed Databases*, H.-J. Schneider, Ed., North-Holland, The Netherlands, pp. 153–184.

LARSON, J. 1983a. Bridging the gap between network and relational database management systems. *Comput.* (Sept.), 82–92.

LARSON, J. 1983b. Granting and revoking discretionary authority. *Inf. Syst. 8*, 4, 251–261.

LARSON, J. 1989. Four reference architectures for distributed database management systems. *Computer, Standards and Interfaces*, Vol. 8, pp. 209–221.

LARSON, J., NAVATHE, S., AND ELMASRI, R. 1989. A theory of attribute equivalence in databases with applications to schema integration. *IEEE Trans. Softw. Eng. 15*, 4 (Apr.), 449–463.

LIEN, Y. 1981. Hierarchical schemata for relational databases. *ACM Trans. Database Syst. 6*, 48–69.

LITWIN, W. 1985. An overview of the multidatabase system MRDSM. In *Proceedings of the ACM National Conference* (Denver, Oct.), pp. 495–504.

LITWIN, W., 1987. The future of heterogeneous databases. In *Proceedings of the Fall Joint Computer Conference* (Dallas, Tex., Oct.), pp. 751–752.

LITWIN, W., AND ABDELLATIF, A. 1986. Multidatabase interoperability. *IEEE Comput. 19*, 12 (Dec.), 10–18.

LITWIN, W., AND ABDELLATIF, A. 1987. An overview of multidatabase manipulation language MDSL. In *IEEE Proc. 75*, 5 (May), 621–632.

LITWIN, W., AND ZEROUAL, A. 1988. Advances in multidatabase systems. In *Research into Networks and Distributed Applications* (Proceedings of the EUTECO '88), Speth, R., Ed. Elsevier Science Publishers B.V., North-Holland, pp. 1137–1151.

LITWIN, W., ABDELLATIF, A., NICOLAS, B., VIGIER, P., AND ZEROUAL, A. 1987. MSQL: A multidatabase language. Tech. Rep. 695, INRIA, BP 105, 78153 Le-Chesnay, France.

LITWIN, W., BOUDENANT, J., ESCULIER, C., FERRIER, A., GLORIEUX, A., LA CHIMIA, J., KABBAJ, K., MOULINOUX, C., ROLIN, P., AND STANGRET, C. 1982. SIRIUS Systems for Distributed Data Management. In *Distributed Data Bases*, H.-J. Schneider, Ed. North-Holland, The Netherlands, pp. 311–366.

LOGAR, T., AND SHETH, A. 1986. Concurrency control issues in heterogeneous distributed database management systems. Tech. Memo, Honeywell Computer Sciences Center, Camden, Minn.

MOTRO, A., AND BUNEMAN, P. 1981. Constructing superviews. In *Proceeding of the ACM SIGMOD Conference* (May), pp. 54–64.

NAVATHE, S., ELMASRI, R., AND LARSON, J. 1986. Integrating user views in database design. *IEEE Comput. 19*, 1 (Jan.), 50–62.

NAVATHE, S., GALA, S., KAMATH, A., KRISHNAMURTHY, A., SAVASERE, A., AND WANG, W. 1989. A federated architecture for heterogeneous information systems. In the *Workshop on Heterogeneous Database Systems* (Chicago, Ill., Dec.).

ONUEGBE, E., RAHIMI, S., AND HEVNER, A. 1983. Local query translation and optimization in a distributed system. In *AFIPS Conference Proceedings*, Vol. 52, *National Computer Conference*, AFIPS Press, pp. 229–239.

OZSU, M., AND VALDURIEZ, P. 1990. *Principles of Distributed Database Systems*. Prentice-Hall, Englewood Cliffs, N.J.

PECKHAM, J., AND MARYANSKI, J. 1988. Semantic data models. *ACM Comput. Surv. 20*, 3 (Sept.), 153–190.

PIATETSKY-SHAPIRO, G., AND JAKOBSON, G. 1987. An intermediate database language and its rule-based transformation to different database languages. *Data and Knowledge Engineering 2*, 1–29.

PU, C. 1987. Superdatabases: Transactions across database boundaries. In *Q. Bull. IEEE-CS TC Data Eng. 10*, 3 (Sept.), 19–25.

RAM, S., AND CHASTAIN, C. 1989. Architecture of distributed data base systems. *J. Syst. Softw. 10*, 2, 77–95.

ROSENTHAL, A., AND REINER, D. 1987. Theoretically sound transformations for practical database design. In *Proceedings of the 6th International Conference on Entity-Relationship Approach* (New York, Nov.), pp. 97–113.

RUSINKIEWICZ, M. 1987. Heterogeneous databases: Towards a federation of autonomous systems. In *Proceedings of the Fall Joint Computer Conference* (Dallas, Tex., Oct.), pp. 751–752.

RUSINKIEWCIZ, M., AND CZEJDO, B. 1987. An approach to query processing in federated database systems. In *Proceedings of the 20th International Conference on System Sciences* (Hawaii, Jan.), pp. 430–440.

RUSINKIEWICZ, M., OSTERMANN, S., ELMAGARMID, A., AND LOA, K. 1990. The distributed operation language for specifying multisystem applications. In *Proceedings of the 1st International Conference on Systems Integration* (Morristown, N.J., Apr.).

RUSINKIEWICZ, M., ELMASRI, R., CZEJDO, B., GEORAKOPOULOUS, D., KARABATIS, G., JAMOUSSI, A., LOA, L., AND LI, Y. 1989. OMNIBASE: Design and implementation of a multidatabase system. In *Proceedings of the 1st Annual Symposium in Parallel and Distributed Processing* (Dallas, Tex., May), pp. 162–169.

RYAN, K., AND LARSON, J. 1986. The use of E-R models in capability schemas. In *Proceedings of the 5th International Conference on the Entity-Relationship Approach* (Dijoin, France, Nov.).

SELINGER, P., AND WADE, B. 1976. An authorization mechanism for a relational database system. *ACM Trans. Database Syst. 1*, 3, 242–255.

SIEGEL, M. 1987. A survey on heterogeneous database systems. Tech. Note 87-174.1, GTE Laboratories, Waltham, Mass.

SHETH, A. 1987a. Heterogeneous distributed database systems: Issues in integration. *The 3rd International Conference on Data Engineering*. IEEE Press, Washington, D.C.

SHETH, A. 1987b. When will we have true heterogeneous database systems? In *Proceedings of the Fall Joint Computer Conference* (Dallas, Tex., Oct.), pp. 747–748.

SHETH, A. 1988a. Managing and integrating unstructured and structured data: Problems of representation, features, and abstraction. In *Proceedings of the 4th International Conference on Data Engineering*. pp. 598–599.

SHETH, A. 1988b. Building Federated Database Systems. In *Distrib. Process. Tech. Comm. Newsl. 10*, 2 (Nov.), 50–58.

SHETH, A., AND GALA, S. 1989. Attribute relationships: An impediment in automating schema integration. In the *Workshop on Heterogeneous Database Systems* (Chicago, Ill., Dec.).

SHETH, A., LARSON, J., AND WATKINS, E. 1988a. TAILOR: A tool for updating views. In

Proceedings of the International Conference on Extending Database Technology (Venice, Italy, Mar.). In *Computer Science*, Vol. 303, Springer-Verlag, New York, pp. 190–213.

SHETH, A., LARSON, J., CORNELLIO, A., AND NAVATHE, S. 1988b. A tool for integrating conceptual schemas and user views. In *Proceedings of 4th International Conference on Data Engineering*. pp. 176–183.

SOUZA, J. 1986. SIS: A schema integration system. In *Proceedings of the BNCOD5 Conference*. pp. 167–185.

TEMPLETON, M., LUND, E., AND WARD, P. 1987a. Pragmatics of access control in Mermaid. In *Q. Bull. IEEE-CS TC Data Eng. 10*, 3 (Sept.), 33–38.

TEMPLETON, M., BRILL, D., CHEN, A., DAO, S., LUND, E., MCGREGOR, R., AND WARD, P. 1987b. Mermaid: A front-end to distributed heterogeneous databases. In *Proc. IEEE 75*, 5 (May), 695–708.

TEOREY, T. 1990. *Database Modeling and Design: The Entity-Relationship Approach*, Chaps. 8–9. Morgan Kaufmann, San Mateo, Calif.

TEOREY, T., YANG, D., AND FRY, J. 1986. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv. 18*, 2 (June), 197–222.

THOMAS, G., et al. 1990. Heterogeneous distributed database systems for production Use. *Comput. Surv. 22*, 3 (Sept.), 237–266.

TSICHRITZIS, D., AND KLUG, A. Eds. 1978. The ANSI/X3/SPARC DBMS framework. *Inf. Syst. 3*, 4.

TSICHRITZIS, D., AND LOCHOVSKY, F. 1982. *Data Models*, Chap. 14. Prentice-Hall, Englewood Cliffs, N.J.

VEIJALAINEN, J., AND POPESCU-ZELETIN, R. 1988. Multidatabase systems in ISO/OSI environment. In *Standards in Information Technology and Industrial Control*, Malagardis, N., and Williams, T., Eds. North-Holland, The Netherlands, pp. 83–97.

WANG, C., AND SPOONER, D. 1987. Access control in a heterogeneous distributed database management system. In *Proceedings of the 6th Symposium on Reliability in Distributed Software and Database Systems* (Mar.).

YU, C., AND CHANG, C. 1984. Distributed query processing. *ACM Comput. Surv. 16*, 4, (Dec.), 399–433.

ZANIOLO, C. 1979. Design of relational views over network schemas. In *Proceedings of the ACM SIGMOD Conference*, pp. 179–190.

BIBLIOGRAPHY

Multi-DBMS and Federated Database Systems: Adiba and Delobel 1977, Adiba and Portal 1978, Belcastro et al. 1988, Bell et al. 1987, Breitbart et al. 1986, Brzezinski et al. 1984, Cardenas 1987, Cardenas and Pirahesh 1980, Chung 1990, Deen et al. 1985, Devor et al. 1982b, Dwyer et al. 1986, Dwyer and Larson 1987, Ferrier and Stangret 1983, Furlani et al. 1983, Ge et al. 1987, Hammer and McLeod 1979, Heimbigner and McLeod 1985, IISS 1986, Jacobson et al. 1988, Landers and Rosenberg 1982, Litwin 1985, Litwin and Vifier 1987, Litwin et al. 1982, Rajinikanth et al. 1990, Rusinkiewicz et al. 1989, Smith et al. 1981, Spaccapietra et al. 1982, Stephenson and Main 1986, Stocker et al. 1984, Templeton et al. 1983, Templeton et al. 1987b, Tsubaki and Hotaka 1980.

Schema Translation: Elmasri et al. 1985, Jajodia and Ng 1983, Kalinichenko 1978, Katz 1980, Klug 1981, Larson 1983a, Lien 1981, Morgestern 1981, Navathe 1980, Navathe and Cheng 1983, Pelagatti et al. 1978, Senko 1976, Sibley and Hardgrave 1977, Shoval and Even-Chaime 1987, Vassiliou and Lachovsky 1980, Zaniolo 1979 (references), Zaniolo 1979, (bibliography).

Schema Integration: Batini and Lenzerini 1984, Batini et al. 1986, Czejdo et al. 1987, Deen et al. 1987, Effelsberg and Mannino 1984, Elmasri 1980, Elmasri et al. 1986, Hayes and Ram 1990, Larson et al. 1989, Mannino and Effelsberg 1984, Navathe et al. 1986, Sheth et al. 1988b, Sheth and Gala 1989, Souza 1986.

Multidatabase Languages: Deen et al. 1987, Jacobs 1985, Lamersdorf et al. 1987, Litwin and Abdellatif 1987, Litwin et al. 1988, Piatetsky-Shapiro and Jakobson 1987, Rusinkiewicz et al. 1989.

Operation Translation and Optimization: Czejdo et al. 1987, Dayal 1983, Deen et al. 1987, Katz 1980, Onuegbe et al. 1983, Vassiliou and Lachovsky 1980, Zaniolo 1979 (bibliography).

Transaction Management: Alonso et al. 1987, Bernstein and Goodman 1981, Breibart et al. 1987, Breitbart and Silberschatz 1988, DE 1987, Du and Elmagarmid 1989, Du et al. 1989, Eliassen and Veijalainen 1987, Elmagarmid and Du 1990, Elmagarmid and Helal 1988, Elmagarmid and Leu 1987, Gligor and Popescu-Zeletin 1985, Logar and Sheth 1986, Pu 1987, Thomson 1987, Veijalainen and Popescu-Zeletin 1986.

ADIBA, M., AND DELOBEL, C. 1977. The problem of the cooperation between different D.B.M.S. In *Architecture and Models in Data Base Management Systems*, Nijssen, G., Ed. North-Holland, The Netherlands.

ADIBA, A., AND PROTAL, D. 1978. A cooperative system for heterogeneous data base management systems. *Inf. Syst. 3*, 209–215.

BATINI, C., AND LENZERINI, M. 1984. A methodology for data schema integration in entity-relationship model. *IEEE Trans. Softw. Eng. SE-10*, 6 (Nov.).

BELL, D., et al. 1987. MULTI-STAR: A multidatabase system for health information systems. In *Proceedings of the 7th International Conference on Medical Informatic* (Rome, Italy, Sept.).

BREITBART, Y., OLSON, P., AND THOMPSON, G. 1986. Database integration in a distributed heterogeneous database system. In *Proceedings 2nd International Conference on Data Engineering.* pp. 301–310.

BREITBART, Y., SILBERSCHATZ, A., AND THOMPSON, G. 1987. An update mechanism for multidatabase systems. In *Q. Bull. IEEE-CS TC Data Eng. 10*, 3 (Sept.), 12–18.

CARDENAS, A., AND PIRAHESH, H. 1980. Data base communication in a heterogeneous data base management system network. *Inf. Syst. 5*, 55–79.

CD 1988. Common Data model*plus. Product literature, Control Data Corp.

CHUNG, C. 1990. DATAPLEX: An access to heterogeneous distributed databases. *Commun. ACM 33*, 1 (Jan.), 70–80.

CODASYL, 1971. Database task group of CODASYL programming language committee. Report. (Apr.).

DAYAL, U. 1983. Processing queries over generalization hierarchies in a multidatabase system. In *Proceedings of the 9th International Conference on Very Large Data Bases.*

DEEN, S., AMIN, R., AND TAYLOR, M. 1987. Data integration in distributed databases. *IEEE Trans. Softw. Eng. SE-13*, 7.

DEEN, S., AMIN, R., OFORI-DWUMFUO, G., AND TAYLOR, M. 1985. The architecture of a generalized distributed data base system-PERCI*. *Comput. J. 28*, 3, 209–215.

DWYER, P., KASRAVI, K., AND PHAM, M. 1986. A heterogeneous distributed database management system (DDTS/RAM). Tech. Rep. CSC-86-7:8216, Honeywell Computer Sciences Center, Camden, Minn.

EFFELSBERG, W., AND MANNINO, M. 1984. Attribute equivalence in global schema design for heterogeneous distributed databases. *Inf. Syst. 9*, 3/4.

ELMAGARMID, A., AND DU, W. 1990. A paradigm for concurrency control in heterogeneous distributed database systems. In *Proceedings of the 6th International Conference on Data Engineering.* (Feb.).

ELMAGARMID, A., AND LEU, Y. 1987. An optimistic concurrency control algorithm for heterogeneous distributed database systems. In *Q. Bull. IEEE-CS TC Data Eng. 10*, 3 (Sept.), 26–32.

ELMASRI, R. 1980. On the design, use, and integration of data models. Ph.D. dissertation, Rep. STAN-CS-80-801, Computer Science Dept., Stanford Univ., Stanford.

FERRIER, A., AND STANGRET, C. 1983. Heterogeneity in the distributed database management system SIRIUS-DELTA. In *Proceedings of the 8th Very Large Data Base Conference* (Mexico City).

FONG, E., AND GOLDFINE, A. 1986. Data base directions: Information resource management: Making it work, Executive Summary. *SIGMOD RECORD 15*, 3 (Sept.).

FURLANI, C., et al. 1983. The automated manufacturing research facility of the national bureau of standards. In *Proceedings of the Summer Computer Simulation Conference* (Vancouver, Canada, July).

HSIAO, D., AND KAMEL, M. 1989. Heterogeneous databases: Proliferation, issues, and solutions. *IEEE Trans. Knowledge Data Eng. 1*, 1, 45–62.

HONG, S., AND MARYANSKI, F. 1988. Database design tool generation via software reusability. In *Proceedings of the COMPSAC.*

JAJODIA, S., AND NG, P. 1983. On representation of relational structures by entity-relationship diagrams. In *Entity Relationship Approach to Software Engineering*, Davis et al., Eds. Elsevier Science Publishers, New York.

KALINICHENKO, L. 1978. Data model transformation method based on axiomatic data model extension. In *Proceedings of the 4th Very Large Data Base Conference.*

KATZ, R. 1980. Database design and translation for multiple data models. Ph.D. dissertation, Memo No. UCB/ERL M80/24, College of Eng., Univ. of California, Berkeley, Calif.

KLUG, A. 1981. Multiple view, multiple data model support in the CHEOPS database management system. Tech. Rep. 418, Computer Science Dept., Univ. of Wisconsin-Madison, Madison, Wisc.

LAMERSDORF, W., ECKHARDT, H., EFFELSBERG, W., JOHANNSEN, W., REINHARDT, K., AND SCHMIDT, J. 1987. Database programming for distributed office systems. In *Proceedings IEEE Comp. Soc. Symp. on Office Automation* (Gaithersburg, Md.).

LARSON, J., NAVATHE, S., AND ELMASRI, R. 1989. A theory of attribute equivalence in databases with applications to schema integration. *IEEE Trans. Softw. Eng. 15*, 4 (Apr.), 449–463.

LITWIN, W., AND VIGIER, P. 1987. New capabilities of the multidatabase system MRDSM. In *HLSUA Forum XLV Proceedings* (New Orleans, Oct.).

MANNINO, M., AND EFFELSBERG, W. 1984. Matching techniques in global schema design. In *Proceedings of First International Conference on Data Engineering* (Apr.), pp. 418–425.

MORGESTERN, M. 1989. A unifying approach for conceptual schema to support multiple data models. In *Proceedings of the 2nd International Conference on Entity-Relationship Approach* (Washington, D.C., Oct.), pp. 281–299.

NAVATHE, S. 1980. An intuitive approach to normalize network structured data. In *Proceedings of the 6th International Conference on Very Large Data Bases.*

NAVATHE, S., AND CHENG, A. 1983. A methodology for database schema mapping from extended entity relationship models into the hierarchical model. In *Entity Relationship Approach to Software Engineering*, Davis, et al. Eds. Elsevier Science Publishers, New York.

PELAGATTI, G., PAOLINI, P., AND BRACCHI, G. 1978. Mapping external views to common data model. *Inf. Syst. 3.*

RAJINIKANTH, M., JACOBSON, G., LAFOND, C., PAPP, W., AND PIATETSKY-SHAPIRO, G. 1990. Multiple database integration in CALIDA: Design and integration. In *Proceedings of the 1st International Conference on Systems Integration* (Apr.).

REINER, D., BROWN, G., FRIEDELL, M., LEHMAN, J., MCKEE, R., RHEINGANS, P., AND ROSENTHAL, A. 1987. A Database designer's workbench. In *Entity-Relationship Approach*, Spaccapietra, S., Ed. Elsevier Science Publishers, New York, pp. 347–360.

SENKO, M., 1976. DIAM as a detailed example of the ANSI SPARC architecture. In *Modelling in Data Base Management Systems*, Nijssen, G., Ed. North-Holland, The Netherlands.

SHIPMAN, D. 1981. The functional data model data language DAPLEX. *ACM Trans. Database Syst. 6*, 1 (Mar.), 140–173.

SMITH, J., et al. 1981. Multibase: Integrating heterogeneous distributed database systems. In *Proceedings of the National Computer Conference.*

SPACCAPIETRA, S., DEMO, B., DILEVA, A., PARENT, C., CELLIS, C., AND BELFAR, K. 1982. An approach to effective heterogeneous database cooperation. In *Distributed Data Sharing Systems*, van de Riet, R., and Litwin, W., Eds. North-Holland, The Netherlands, pp. 209–218.

STEPHENSON, G., AND MAIN, R. 1986. ARCHEDDA Prototype. Final Report. CRI, Great Britain.

STOCKER, P., et al. 1984. Proteus: A heterogeneous distributed data-base project. In *Databases: Role and Structure*, Gray, P., and Atkinson, M., Eds. Cambridge University Press, New York.

TEMPLETON, M., BRILL, D., CHEN, A., DAO, S., AND LUND, E. 1986. Mermaid: Experiences with network operation. In *Proceedings of the 2nd International Conference on Data Engineering.*

TEMPLETON, M., BRILL, D., HWANG, A., KAMENY, I., AND LUND, E. 1983. An overview of the mermaid system. A frontend to heterogeneous databases. In *Proceedings of EASCON 83.*

THOMSON, G. 1987. Multidatabase concurrency control. Ph.D. dissertation, Oklahoma State University.

TSUBAKI, M., AND HOTAKA, R. 1980. Distributed multidatabase environment with a supervisory data dictionary database. In *Entity-Relationship Approach to System Analysis and Design*, Chen P., Ed. North-Holland, The Netherlands.

VASSILIOU, Y., AND LOCHOVSKY, F. 1980. DBMS transaction translation. In *Proceedings COMPSAC 80, IEEE Computer Software and Application Conference.*

VEIJALAINEN, J., AND POPESCU-ZELETIN, R. 1986. On multi-database transactions in a cooperative, autonomous environment. Tech. Rep., Hahn-Meitner Institut, Berlin GmnH, D-1000 Berlin 39, FRG.

ZANIOLO, C. 1979. Multimodel external schemas for CODASYL data base management systems. In *Data Base Architecture*, Bracchi, G., and Nijssen, G., Eds. North-Holland, The Netherlands.

GLOSSARY

Accessing Processor: Software that accepts commands and produces data by executing commands against a database.

Class of Users: A set of users performing closely related tasks.

Common Data Model (CDM): A data model to which schemas of different component DBMSs are translated for the purpose of representation in a common format and facilitation of schema integration.

Component Database Administrator (component DBA): The administrator of a component DBS (also called local DBA) who, among other things, decides data access rights of local users as well as (individuals and/or classes) of federation uses. It is the component DBA's responsibility to manage the local schema, translate it to create the component schemas, and define export schemas.

Component DBMS: A DBMS participating in a multidatabase system. A component DBMS participating in an FDBS is autonomous and allows local operations as well as global (federation) operations that meet its constraints.

Component Schema: A translation of a local schema into an equivalent schema in the common data model.

Constructing Processor: Software that partitions and/or replicates operations produced by a single processor for execution by two or more processors. Also software that merges data produced by two or more processors for consumption by another processor.

Database Management System (DBMS): Software that manages a collection of structured data. Management includes providing data management

services including data access, constraint enforcement, and consistency management.

Database System (DBS): A database system (DBS) consists of a DBMS that manages one or more databases.

Distributed DBMS: A system that manages multiple databases.

Export Schema: A subschema of a component schema specifically defined to express the access constraints on a class of federation users to access the data represented in the component schema.

External Schema: A subschema or a view defined over a federated schema primarily for a pragmatic reason of not having to define too many federated schemas or to tailor a federated schema for smaller groups of federation users than that of a federated schema.

Federated Database System (FDBS): A system that is created to provide operations on databases managed by autonomous, and possibly heterogeneous, DBSs. The software that manages an FDBS is called a federated DBMS (FDBMS).

Federated Schema: An integration of several export schemas created to represent data access requirements and rights of a class or group of federation users.

Federation Administrator (federation DBA): The administrator of a federated schema or the federated database system who, among other things, creates and maintains federated and external schemas.

Federation User: A user of an FDBS or an application running over an FDBS.

Filtering Processor: Software that constrains operations that can be applied or data that can be passed to another processor.

Global (or External) Operation: An operation that is not submitted by a local user (e.g., an operation submitted to the component DBS by the FDBMS or another component DBS).

Local Schema: A schema of a component DBS in a native data model of the component DBMS.

Local User: A user of a component DBS.

Multidatabase System (MDBS): A system that allows operations on multiple databases.

Processor: Software that performs operations on data and commands.

Schema Object: A description of a data element in a schema. For example, a schema expressed in the Entity Relationship model has three types of schema objects: entity types, relationship types, and attribute definitions. An example of an entity type schema object is an "Employee" entity type.

Schema, Subschema, and View: A representation of the structure (syntax), semantics, and constraints on the use of a database (or its portion) in a particular data model. A schema is a collection of schema objects. A subschema is a collection of subsets of that schema's objects. A view is any connected portion of a schema. In other words, a schema is a collection of views.

Transforming Processor: Software that translates commands from one language or format to another language or format or translates data from one format to another.

User: An individual or an application using a database system.

Appendix: Features of Some FDBS/Multi-DBMS Efforts

Table A.1 summarizes the choice of a common data model and the types of component DBMSs integrated into some of the experimental prototype systems. Table A.2 gives an empirical and partial list of significant or interesting features of some of these efforts. No effort is made to present all systems that exhibit FDBS features or to capture all important details of those mentioned.

Table A.1. Data Models of Various Multi-DBMS/FDBS Efforts

Effort	CDM	Types of Component DBMSs	Reference
ADDS	Extended Relational	Hierarchical, Relational, Files	[Breitbart et al. 1986]
CALIDA	Relational like	Relational, Relational like, Files	[Jacobson et al. 1988]
DDTS	Relational[a]	Network, Relational	[Dwyer and Larson 1987]
DQS	Relational	Hierarchical, Network, Relational	[Belcastro et al. 1988]
IISS	E-R Variant (IDEF1x)	Network, Relational	[IISS 1986]
Mermaid	Relational (DIL)	Relational (partial integration of text)	[Templeton et al. 1987b]
MRDSM	Relational	Relational	[Litwin 1985]
Multibase	Functional (DAPLEX)	Network, Relational	[Landers and Rosenberg 1982]
OMNIBASE	Extended Relational	Relational	[Rusinkiewicz et al. 1989]
SIRUS-DELTA	Relational	Network, Relational	[Litwin et al. 1982]
SCOOP	E-R	Hierarchical, Network, Relational	[Spaccapietra et al. 1982]

[a] Extended E-R is used for integrity constraint enforcement at the federated schema level and at the external schema level. The primary CDM can be said to be relational since the internal command language is based on the relational algebra.

Table A.2. Significant Features

Effort	Significant Features
ADDS	Tightly coupled, limited updates, in-house use of the prototype system
CALIDA	More like loosely coupled, intermediate language, interactive, menu-driven user interface, data dictionary editor/browser, in-house use of the prototype system
DDTS	Tightly coupled architecture, integrity constraint checking, local query optimization, use of local and long haul communication
DQS	Tightly coupled, completeness of implementation, auxiliary schema, IBM environment, attention to autonomy
IISS	More like tightly coupled, queries must be compiled, forms-based user interface
Mermaid	Tightly coupled, completeness of implementation, data dictionary, access control, global query optimization, extensions to include texts, experience with communication systems, workstation environment
MRDSM	Loosely coupled architecture, multidatabase language, dealing with multiple semantic interpretations
Multibase	Tightly coupled, completeness of implementation, schema integration, global query optimization, auxiliary schema, architecture, number of component DBMSs integrated in various prototypes
OMNIBASE	Loosely coupled, DEC environment, query processing, distributed DBMS as a component DBMS

Received November 1988; final revision accepted June 1990.