1992-05

Federated database management system: Requirements, issues and solutions

Kamel, Magdi N.

Database Management, Computer Communications, Vol. 15, No. 4, May 1992

http://hdl.handle.net/10945/46791

Federated database management system: Requirements, issues and solutions

Magdi N Kamel and Nabil N Kamel* discuss the requirements and objectives of a federated DBMS

The use of database management systems (DBMS) to replace conventional file processing systems has dramatically increased in the past years. Although the use of DBMSs overcomes many of the limitations of file processing systems, many important applications require access to and integration of information among several and often incompatible DBMSs. In this paper we discuss an approach, known as the federated database approach, that allows users and applications to access and manipulate data across several heterogeneous databases while maintaining their autonomy. We discuss the requirements and objectives of a federated database management system, and outline the major issues and challenges for building and using such a system. In particular, we address the design issues from three angles: transaction management, system architecture, and schema integration. Also, we present a five-step integration methodology followed by a comprehensive example to illustrate the concepts and techniques involved in this integration methodology.

Keywords: database management systems, federated databases, transaction management, system architecture, schema integration

Department of Information Systems, Naval Postgraduate School, Monterey, CA 93943, USA. *Computer and Information Sciences Department, University of Florida, Gainsville, FL 32611, USA

The use of database management systems (DBMS) to replace conventional file processing systems has dramatically increased in the past years. Today, a typical organization maintains numerous separate and different DBMSs, databases and their applications. Although the use of DBMSs overcomes many of the limitations of file processing systems, organizations soon discover the need to access and share data among different databases for decision support, overall control, corporate assessment, and high level planning. This need is certain to accelerate with the globalization of business, whereby the scope and presence of organizations expand beyond their geographical boundaries[1].

This situation has led to the emergence of the *heterogeneous distributed database* scenario. In this scenario, a variety of large and small computers, each with its own autonomous and often incompatible DBMS, may be tied together in a network. This network could consist of local area, wide area and long-haul networks. Under current technology, however, a user accessing any database in this network must abide by the syntactic and semantic rules of that database. Developing an application that requires global access to data maintained by these separate databases is quite difficult. First, the databases that contain the data to be accessed are identified. Second, several queries in different languages are formulated and executed on different computers. Third, the results are transferred to the requesting site. Fourth, the results of the different sites are combined. Finally, the result of the original request is extracted and formatted.

0140-3664/92/004270-09 © 1992 Butterworth-Heinemann Ltd

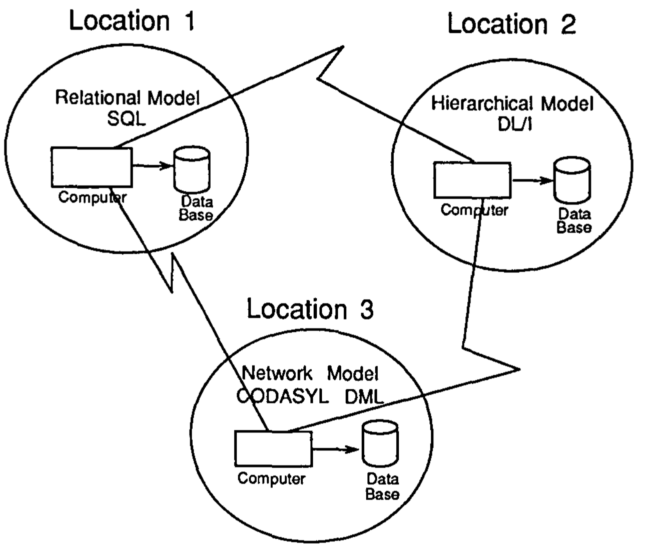

Figure 1. Heterogeneous distributed database management system - a scenario

AIRCRAFT_CLASS (<u>DESIGNATION</u>, TYPE, LENGTH, WINGSPAN, SPEED, RANGE)
AIRCRAFT (<u>SER#</u>, DESIGNATION, CONFIGURATION)

Figure 2. Relational schema at location 1

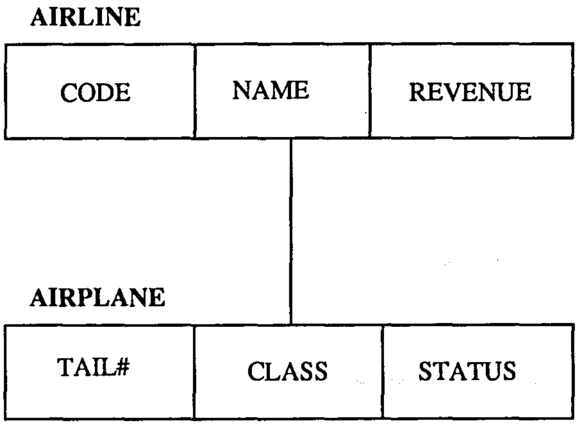

Figure 3. Hierarchical schema at location 2

As an example, consider the heterogeneous database depicted in Figure 1. In this depiction, three different computers, tied together in a network, maintain three different databases. At location 1, there is a database managed by a relational DBMS. At location 2, there is a second database managed by a hierarchical DBMS while at location 3, a third database is managed by a network DBMS.

The relational database at location 1 maintains information on aircraft classes (ex. DC-10) and individual aircrafts of each class. The hierarchical database at location 2 maintains information on airlines and their respective airplanes. The local schemas of these databases are shown in Figures 2 and 3, respectively.

To access the hierarchical database at location 2, a user at location 1 or 3 must be familiar with or learn the hierarchical data model. Similarly, a user at location 2 or 3 must be versed in the relational model to access the relational database at location 1. An application requiring data from two or more locations (for example, data that relate airlines and aircraft classes) cannot be answered easily in this environment.

To increase the degree of access and sharing among the heterogeneous databases and allow applications that access global data, a *federated database* approach has been originally proposed by Hammer and Mcleod[2]. Under this approach each local database is considered a logical component in the federation. These components are tied together by one or more federal schemas that represent the integration of several local schemas. The federal schemas represent, therefore, information that can be shared by the federation components. The software that provides control and coordination of the component databases is known as a *federated database management system* (FDBMS)[3] (see Figure 4).

In the next section we discuss the main requirements for a federated database management system, and then outline the different issues and design alternatives associated with this approach. An illustrative example is given, and finally we conclude with a summary and indicate directions for the future.

FEDERATED DBMS REQUIREMENTS

In this section we present the main requirements and objectives that we feel should be met in a federated database management system (see also the discussion by Kim[4]).

1. The user should be able to access a number of heterogeneous databases as if accessing a single database

As indicated in the previous section, this is a main objective of a federated database management system. The system should allow the definition of one or several global schemas that represent the integration of individual local schemas in the network. By issuing queries on a global schema, a user can access a set of heterogeneous databases as if he is accessing a single local database. In other words, a FDBMS should provide distribution transparency for the data being accessed.

2. The user should be able to access any database using a familiar data model and language

In addition to distribution transparency, a FDBMS should also provide for heterogeneity transparency. This means that a FDBMS should hide the heterogeneity of different data models and languages. A user should be able to access other schemas in exactly the same way he is

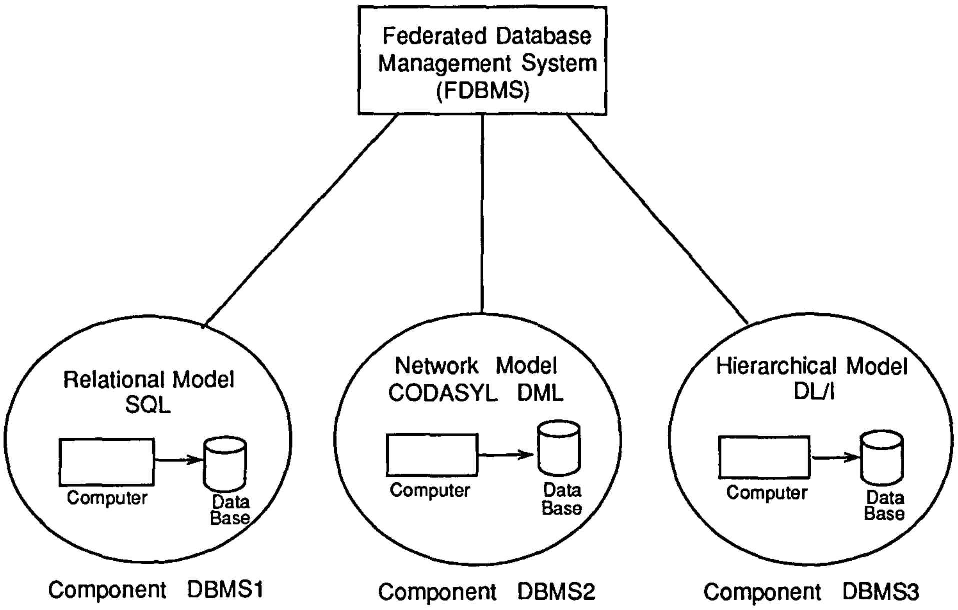

Figure 4. Federated database management system and its components

accessing his local database using a familiar model and language. Users should not be required to learn new data models and languages to access other databases.

3. FDBMS should not require any significant changes to existing database systems or applications
One of the major reasons behind the federated database approach is to preserve the huge investment in existing systems. This investment consists of the database management systems, the databases, and the database applications. A FDBMS should, therefore, not require any significant changes to existing databases and applications. Any changes that might be required to database management systems should be done in an incremental and isolated fashion.

4. The system should accommodate the addition of new databases to the network
Since there are many types of database systems, and since organizations continually introduce new DBMS types to meet their application requirements, a FDBMS should be extensible. This means that a FDBMS should accommodate the addition of new DBMS and their databases to the network easily and with minimal changes.

5. The user should be able to access the databases for both retrievals and updates
Users should be able to perform all types of operations, including updates, on the global schema. To support updates, both transactional integrity, which refers to the consistency of the database in the presence of concurrent access, and semantic integrity, which refers to the consistency of the database with respect to integrity constraints, needs to be maintained. Maintaining these types of integrity in a federated environment is significantly more difficult than in a homogeneous environment.

6. Performance of FDBMS should be comparable to that of homogeneous distributed systems
Hiding the heterogeneity of the databases in the network requires adding several layers of processing which would considerably increase the overhead of FDBMS. To be practical, the performance of a FDBMS should be at least comparable to that of a homogeneous distributed system. This is a difficult yet an essential requirement for FDBMS.

We feel that we should strive to build a FDBMS that meets all of the above requirements. Current research prototypes satisfy some but not *all* of these requirements. Examples of these prototype systems include DDTS[5], HD-DBMS[6], Mermaid[7], MULTIBASE[8] and SERIUS-DELTA[9].

FEDERATED DBMS DESIGN ISSUES

The functionality required for a FDBMS presents the system designer and the system integrator with several complex design alternatives and integration challenges. We classify these alternatives and challenges under three categories: Transaction management, architecture and schema integration. These issues are outlined below.

Transaction management issues

We divide transaction management issues into query processing and update processing issues.

Query processing
To allow users to pose queries on a global schema, an additional control component, known as the *global* or *federal controller*, is required. The global controller

maintains the definition of the global schemas and acts as a coordinator and translator: it receives a global query in a user specific language; translates it into an equivalent query on a common-model global schema; decomposes and translates the common-model query into subqueries that operate on the local schemas; sends the subqueries to the corresponding local database sites for processing; collects and reformats the result; and sends it back to the originating site. This process is summarized in Figure 5. Dayal[10] provides detailed information on query processing in heterogeneous database environments.

Update processing

There are four issues associated with update processing: global concurrency control, global deadlock handling, global data recovery, and global semantic integrity enforcement. We briefly discuss each of these issues.

Global concurrency control: to ensure consistency of transactions that reside on several sites, global concurrency control is needed to ensure that transactions execute in a *serializable* manner[11]. There are two problems with enforcing global concurrency control. First, different DBMSs use different concurrency control methods to ensure serializability of local transactions. While the majority of commercial systems employ two-phase locking, some may employ different techniques such as timestamping or optimistic methods. Additionally, the implementation of the same method of concurrency control may differ from one system to another. For example, the granularity of locking may be at the file level in one system and at the record level in another. Second, the DBMS at the various sites were not designed to communicate with each other to coordinate their activities. To accomplish global control, the FDBMS must ensure that concurrency control is performed globally by preventing, for example, a local DBMS from releasing locks until updates at all other locations are complete. Global serializability, however, provides a low degree of concurrency, and therefore might be too strong a requirement and alternative paradigms might be needed[12].

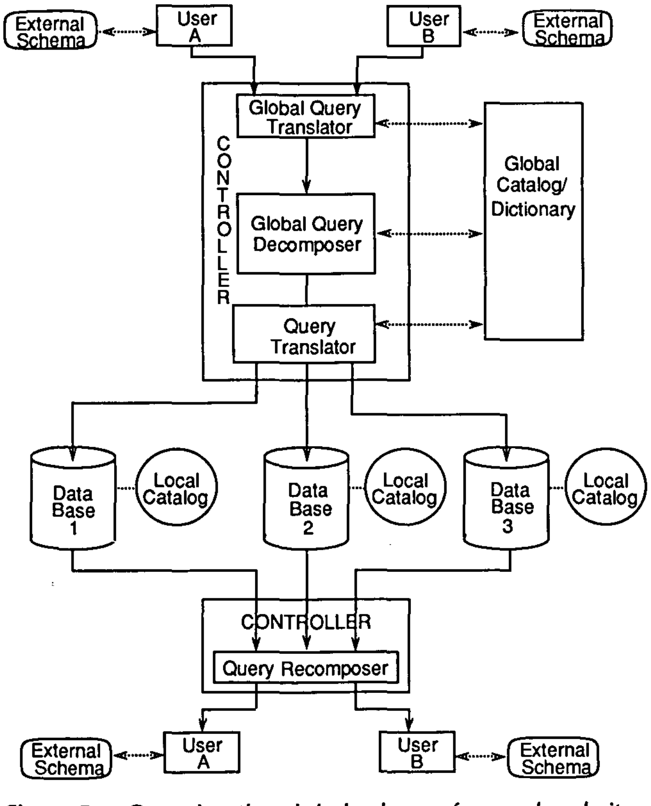

Figure 5. Querying the global schema from a local site

Global deadlock handling: a second problem which must be handled by the FDBMS is global deadlock detection. A deadlock is a situation when each of two transactions is waiting for the other to release locks on an item. Global deadlock detection is inherently a difficult problem for several reasons. First, a local process of a global transaction has no knowledge of the non-local portions of the transaction. Second, a global process has no knowledge of the local transactions. A *global wait-for graph* needs to be constructed and analysed for the existence of cycles to discover deadlocks. Unfortunately, the construction of a global wait-for graph requires complex algorithms[13]. There is a general agreement that the most practical solution is to use timeouts for global deadlock detection.

Global data recovery: to ensure global update atomicity, a global recovery method is needed. The most commonly used method in distributed environments is the Two-Phase Commit (2PC) protocol. The 2PC consists of two phases. In the first phase a coordinator asks all participant DBMS to prepare to commit; each participant answers READY if it is ready to commit, or ABORT otherwise. If all participants have answered READY, the coordinator decides to commit the transaction. If some of the participants has answered ABORT, it decides to abort the transaction. In the second phase, the coordinator informs all the participants of its decision, which is implemented by the participants. The problem of implementing the two-phase commit in a heterogeneous autonomous environment is that local DBMSs do not have the capabilities to inform the FDBMS of the prepare to commit state. Current DBMSs do not provide an external interface to perform the prepare to commit phase. This is an area where future standardization could be beneficial to facilitate integration.

It should be noted that the 2PC protocol automatically satisfies global concurrency control, since the 2PC forces local DBMSs to commit and release locks only after all other DBMSs involved in the global update are prepared to commit.

Global semantic integrity enforcement: one way to prevent inaccuracies from being introduced in a database is to enforce *semantic integrity constraints.* These constraints are predicates that define consistent database states. A semantic integrity system is responsible for ensuring the consistency of the database by rejecting updates that would lead to an inconsistent state[14]. The problem of global semantic integrity enforcement is intrinsically a problem of global transaction management, since enforcing an integrity constraint may require access to data at different sites. This is a complex and difficult area where very little work has been done.

Architectural considerations

Several design alternatives must be addressed at this level. First, the number and organization of the global schemas. This could range from a single schema to several global schemas arranged in a complex structure. Second, the location of the controller. The controller could reside on a special node, on an existing node, or could be distributed among all (or some) nodes of the network. Third, the additional facilities that might be required for the local database to interact with the global environment. This could include concurrency control mechanisms to allow the simultaneous access of data if this mechanism is absent at the local level. It could also include a communication mechanism in the local sites to communicate with the controller.

The choice of a particular architecture will depend on many factors. These include the ease of changing and maintaining the global schema, system performance, system complexity, availability and reliability. Mcleod and Heimbigner[15] provide an extended discussion on the design issues of distributed architectures.

Schema integration considerations

A major challenge for integrating existing databases is the construction of a global unified schema that represents the integration of local schemas. Several problems arise in schema integration in this environment due to the structural and semantical differences of the schemas to be merged. First, schemas at different locations are represented in different data models. This situation requires the use of a common data model to interconnect these diverse data models. Second, many conflicts may arise when integrating different schemas. These conflicts include:

- *Name conflicts,* which involve synonyms, different names for a construct representing the same fact, and homonyms, different facts represented by the same construct name;
- *Different representation conflicts,* which arise when the same facts are represented by different constructs in different schemas. For example, a real-world entity is represented as an entity in one schema and as an attribute in another schema;
- *Conflicts in application semantics,* which result due to different perception by different users. For example, in one schema a relationship between two entities is characterized as one to one, while it is characterized as one to many in another schema.

Conflict identification and resolution is, therefore, crucial to the problem of integration. Third, when integrating different schemas, hidden relationships that do not appear in the individual schemas need be discovered. For a general survey on view integration methodologies, see Batini *et al.*[16].

In the next section we present a five step methodology for schema integration in a heterogeneous environment. The methodology captures the essence of various proposals in the literature. The objective of this methodology is to provide the database integrator with a systematic approach and an understanding of the issues involved for successful integration. A comprehensive example follows to illustrate the implementation of the proposed methodology.

Five-step integration methodology

Step 1. Formulation of an integration policy
A policy of integration needs to be formulated before integration can take place. This policy includes deciding upon the subset schema that each site is willing to share with other sites, known as *export schema,* and the tailored, integrated global view for each site.

These policy decisions will normally be made at a high level of the organization with close interaction with the database administrator at each site.

Step 2. Schema transformation
Once a policy of integration is formulated and an export schema for each site is agreed upon, each local schema is translated into an equivalent schema in an intermediate common data model. This resulting schema is known as the *common-model local schema.* Subsequently, the export schema is specified as a subschema of the common-model local schema.

The common data model should be semantically rich enough to subsume all local data models. Object-oriented data models have recently been proposed as good candidates for a common data model. They include all key data modelling concepts found in current database systems and the newly emerging object-oriented database systems.

Step 3. Conflict identification
In this step, individual schemas are analysed and compared to identify possible conflicts. A simple classification of such conflicts was presented in the previous section. It is also during this step that inter-schema relationships are identified.

Step 4. Conflict resolution
Once conflicts are identified, attempts are made to resolve them. It is during this step that the user feedback is crucial to clarify the semantics of each schema.

Step 5. Schema merging
This step involves merging export schemas of individual sites into a global schema. The resulting schema is

examined and, if necessary, restructured so that it has these desirable qualities[16]:

- *Completeness and correctness:* the resulting schema must represent all the properties of the underlying export schemas correctly;
- *Minimality:* concepts in the global schema should not be duplicated;
- *Understandability:* the global schema should be easily understandable by both the users and the designers.

The development process of this methodology is represented by the multilevel architecture of Figure 6.

COMPREHENSIVE EXAMPLE

Consider the heterogeneous database scenario of Figure 1 and a requirement to integrate the relational and hierarchical schemas at locations 1 and 2. This requirement stemmed from the desire to increase data sharing and allow queries to span both schemas. For example, queries that relate airlines and aircraft classes.

Step 1. Formulation of an integration policy

For simplicity, this example assumes that the integration policy defines the entire schemas at locations 1 and 2 as the export schema of each location. In other words, each location is willing to share its schema in its entirety. Generally, each site may wish to share only a subset of its schema with other users. It is also assumed that after integration, each location will have the total integrated schema as its tailored global view rather than a subset of it.

Step 2. Schema transformation

In this step, each local schema is translated into a common-model schema and an export schema is defined on the common-model schema. We assume the entity relationship model (ERM) as the intermediate common-model. A simplified symbol set of this model is shown in Figure 7.

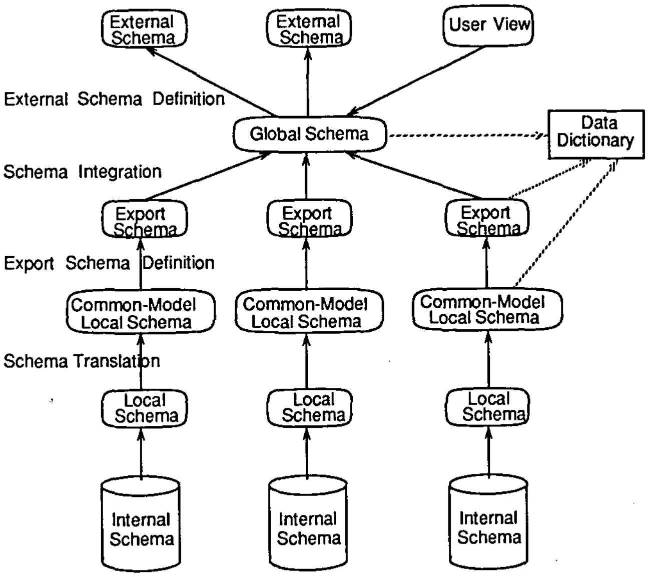

Figure 6. Architecture for developing a federated database system

The relational and hierarchical schema, shown in Figures 2 and 3, are transformed into their equivalent ERM schema, as depicted in Figures 8 and 9.

Step 3. Conflict identification

By examining the export schemas at locations 1 and 2, shown in Figures 8 and 9, in preparation for merging, we discover several conflicts. First, the entity AIRCRAFT in the first schema and the entity AIRPLANE in the second schema correspond to the same concept. Similarly, the attributes SER# and TAIL# in the first and second schema, respectively, are equivalent. Second, Aircraft Class is represented in the two schemas differently: it is an entity in the first schema and an attribute in the second. The reason for having this different representation comes from the different relevance that Aircraft Class has in the two schemas. These conflicts need to be resolved before a global schema can be constructed.

Step 4. Conflict resolution

This step resolves the conflicts identified in the previous step. First, names of the same concept should be unified

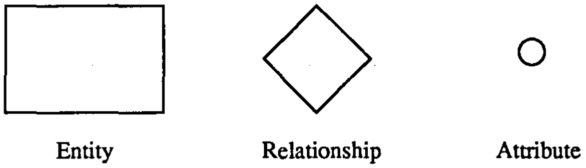

Figure 7. Entity Relationship Model symbols

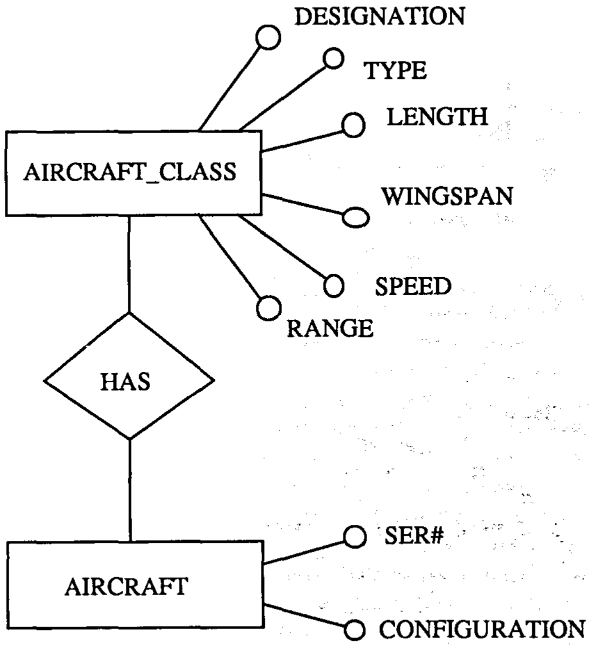

Figure 8. Equivalent ERM schema of relational schema at location 1

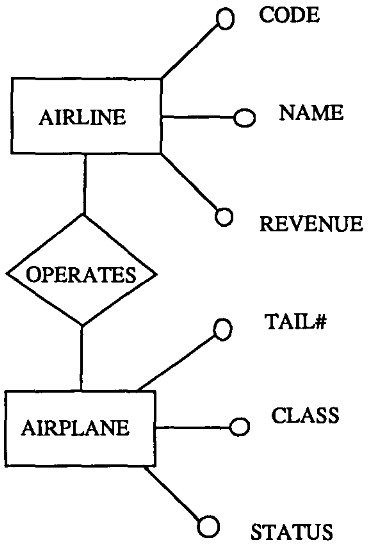

Figure 9. Equivalent ERM schema of hierarchical schema at location 2

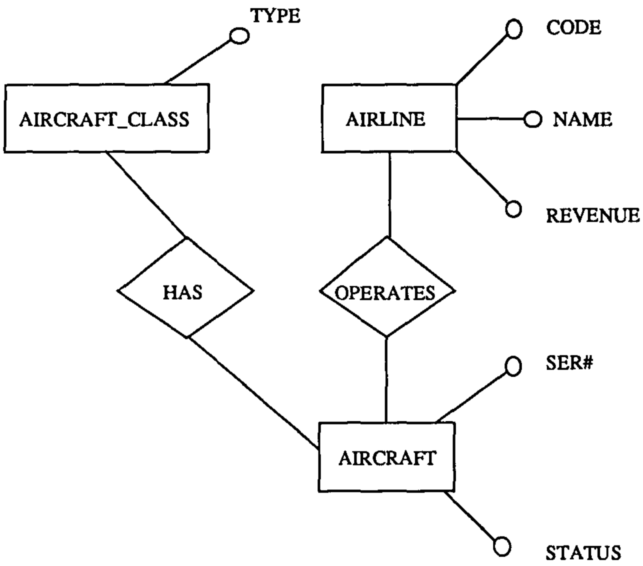

Figure 10. Modified ERM schema of hierarchical schema at location 2

into a single name. We do this by changing the names of the entity AIRPLANE and attribute TAIL# in the second schema into AIRCRAFT and SER#, respectively. Second, we have to conform the different representations of Aircraft Class in the two schemas. We accomplish this by transforming the attribute Class into an entity in the second schema and add a new attribute, Type, to it. These changes are reflected in the modified schema at location 2, depicted in Figure 10.

Step 5. Schema merging

This is the final step in the integration process whereby the local export schemas are integrated in a unified common-model global schema. The unified global ERM schema for this example is shown in Figure 11.

This common-model unified global schema is mapped to the appropriate data model for each user. For example, a user of the relational database at location 1 views the unified global schema as the relational schema of Figure 12. Similarly, a user of the hierarchical database at location 2 will view the unified global schema as a hierarchical schema.

Now, consider a query issued at location 1 on the global schema. This query requests the serial number and status of Egypt Air aircrafts with a range greater than 1000 miles. The formulation of this query using the relational

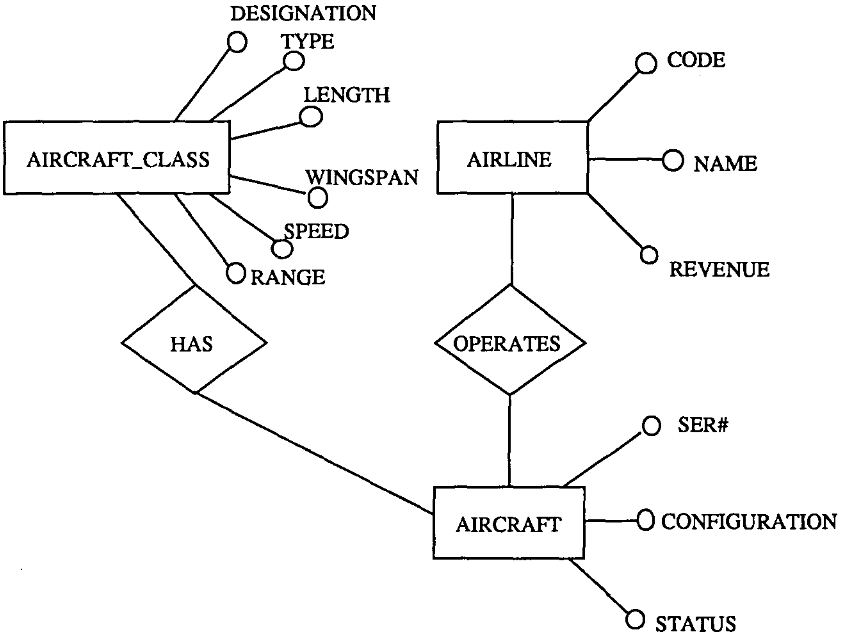

Figure 11. Global ERM schema of local schemas at locations 1 and 2

```
AIRCRAFT_CLASS (DESIGNATION, TYPE, LENGTH,
WINGSPAN, SPEED, RANGE)
AIRLINE (CODE, NAME REVENUE)
AIRCRAFT (SER#, DESIGNATION, CONFIGURATION)
```

Figure 12. Relational schema equivalent of the global schema in Figure 11

language SQL of a user at location 1 is shown in Figure 13.

The controller translates this query into an equivalent one on the global intermediate schema of Figure 11. It then decomposes and translates the resulting query into two queries that operate on the local schemas at locations 1 and 2. The first one, shown in Figure 14a, is written in SQL and operates on the relational database at location 1. The second query, shown in Figure 14b, is written in DL/I and operates on the hierarchical database at location 2. The results of queries 14a and b, saved in $STEMP and $TTEMP, respectively, are sent back to the controller where they are joined on serial (tail) number to produce the answer to the query. This result is reformatted and sent to the originating site, location 1.

As indicated earlier, a user gets the results of his query without being aware of the location, distribution, or heterogeneity of the databases being accessed.

SUMMARY AND CONCLUSIONS

Information integration will be a key factor in the survival and prosperity of organizations in the 1990s[17]. Many important applications will require access and sharing of data among multiple heterogeneous databases. Regardless of their size and structure, organizations will increasingly look to information technology to integrate their numerous existing information systems.

In this paper we have discussed an approach, known as the *federated database* approach, that allows users and aplications to access and manipulate data across several heterogeneous databases as if they are accessing a single database while maintaining the autonomy of the databases in the network. We present our view of what the requirements and objectives of a federated database management system should be, and outline the major issues and challenges that face the designers and users for building and using such a system. A comprehensive example was used to illustrate the concepts and techniques involved in integrating databases in a federated environment. We believe that federated database management systems will be a major trend for distributed data processing in the years to come.

```
SELECT SER#, STATUS
FROM AIRCRAFT_CLASS, AIRLINE, AIRCRAFT
WHERE AIRCRAFT_CLASS.DESIGNATION = AIRCRAFT.
DESIGNATION AND
      AIRCRAFT.CODE = AIRLINE.CODE AND
      NAME = "EGYPT AIR" AND
      RANGE>1,000
```

Figure 13. Relational query on the global schema

```
SELECT SER# INTO $STEMP
FROM AIRCRAFT_CLASS, AIRCRAFT
WHERE AIRCRAFT_CLASS.DESIGNATION = AIRCRAFT.
      DESIGNATION AND RANGE>1,000
```

```
GU AIRLINE(NAME = "EGYPT AIR")
DO WHILE data remains
      GNP AIRPLANE
      write TAIL#, STATUS into $TTEMP
END-DO
```

Figure 14. Decomposition of the global query into local queries. Local query on (a) relational database at location 1; (b) hierarchical database at location 2

REFERENCES

1 **Madnick, S E and Wang, Y R** 'CISL: Composing answers from disparate information systems' *Position Papers of 1989 Workshop on Heterogeneous Databases,* Evaston, IL, USA (December 1989)

2 **Hammer, M and McLeod, D** *On the architecture of database management systems,* Computer Science Department Technical Report 79-4, University of Southern California, USA (April 1979)

3 **Sheth, A P and Larson, J A** 'Federated database systems for managing distributed, heterogeneous, and autonomous databases' *ACM Comput. Surv.,* Vol 22 No 3 (September 1990) pp 183–236

4 **Kim, W** 'Research directions for integrating heterogeneous databases' *Position papers of 1989 Workshop on Heterogeneous Databases,* Evanston, IL, USA (December 1989)

5 **Dwyer, P and Larson, J** 'Some experiences with a distributed database testbed system' *Proc. IEEE,* Vol 75 No 5 (May 1987) pp 633–648

6 **Cardenas, A** 'Heterogeneous distributed database management: The HD-DBMS' *Proc. IEEE,* Vol 75 No 5 (May 1987) pp 588–600

7 **Templeton, M, Brill, D, Chen, A, Dao, S and Lund, E**'Mermaid – Experiences with network operation' *Proc. 2nd Int. Conf. on Data Engineering,* Los Angeles, CA, USA (February 1986) pp 292–300

8 **Landers, T and Rosenberg, R** 'An overview of MULTIBASE', in **Schneider, H J (ed)** *Distributed Databases,* North Holland, Netherlands (1982) pp 153–183

9 **Ferrier, A and Stangret, C** 'Heterogeneity in the distributed database management system SIRIUS-DELTA' *Proc. 8th Int. Conf. Very large DataBases,* Mexico City, Mexico (1982)

10 **Dayal, U** 'Processing queries over generalization hierarchies in a multidatabase systems' *Proc. 9th Int.*

Conf. Very large Databases, Florence, Italy (1983)

11 **Bernstein, P, Hadzilacos, V and Goodman, N** *Concurrency Control and Recovery in Database Systems,* Addison-Wesley, New York (1987)

12 **Du, W and Elmagarmid, A K** 'Quasi serializability: A correctness criterion for global concurrency control in InterBase' *Proc. 15th Int. Conf. Very Large Databases,* Amsterdam, Netherlands (1989) pp 347-355

13 **Obermarck, R** 'Distributed deadlock detection algorithm' *ACM Trans. Database Syst.,* Vol 7 No 2 (June 1982)

14 **Kamel, M and Davidson, S** 'Redundancy: an approach to the efficient implementation of semantic integrity assertions' *Proc. 23rd Ann. Int. Conf. on Syst. Sci.,* Kailua-Kona, HI, USA (January 1990) pp 393-399

15 **Mcleod, D and Heimbigner, D** 'A federated architecture for database systems' *Proc. Nat. Comput. Conf.,* (1980) pp 283-289

16 **Batini, C, Lenzerini, M and Navathe, S** 'A comparative analysis of methodologies for database schema integration' *ACM Comput. Surv.,* Vol 18 No 4 (December 1986) pp 323-364

17 *The Landmark MIT Study: Management in the 1990s,* **Arthur Young,** UK (1989)