Edgar H. Sibley
Panel Editor

Diverse database management systems are used in large organizations. The heterogeneous distributed database system (DDS) can provide a flexible integration of diverse databases for users and applications. This is because it allows for retrieval and update of distributed data under different data systems giving the illusion of accessing a single centralized database system.

DATAPLEX: An Access to Heterogeneous Distributed Databases

CORRECTION: THE ORDER OF ALL FIGURES WAS ERRONEOUSLY REVERSED. PLEASE REFER TO THE CORRIGENDUM IN COMMUNICATIONS OF THE ACM 33, 4 (APRIL), P. 459.

Chin-Wan Chung

In a typical data-management environment of the manufacturing industry, there are a number of engineering, manufacturing, and business data centers run by various geographically dispersed units. The choices of database management systems (DBMSs) by these data centers are diverse. Furthermore, there can be several different DBMSs in a data center.

Currently, there are no effective means to share these heterogeneous databases. The lack of effective data sharing causes inefficient engineering and manufacturing activities and business operations. Duplicated data at different locations results in data inconsistency. The development of the same applications in different data manipulation languages used by different DBMSs incurs unnecessary human cost.

One way to solve this problem is to standardize on a single DBMS. In a large organization, however, it is difficult to standardize on a single DBMS for the following reasons:

- The requirements of the DBMS from different units are diverse. Some DBMSs are better than others in processing a particular class of applications. There is no single DBMS that satisfies all these requirements.
- Each DBMS can run only on certain computers.
- Transition from one DBMS to another is extremely expensive because of database and application conversion. Initial standardization requires a high conversion cost. Also, technology is rapidly changing. If a new standard DBMS is chosen to replace an old one in order to take advantage of new technology, all the data and applications in the organization have to be converted.

Consequently, coexistence of many different DBMSs is a way of life in large organizations. In light of diverse DBMSs and the existence of cooperating autonomous components in an organization, the heterogeneous DDS is an effective means of sharing data. The heterogeneous DDS can facilitate the following:

- Each data center can select the best DBMS for its environment because the heterogeneous DDS allows the co-existence of different DBMSs.
- When a DBMS of a data center is obsolete, the DBMS can be changed without impacting other data centers
- When a DBMS has to be changed, the old DBMS and the new one can communicate as members of the heterogeneous DDS until the conversion process is complete. This allows a phased conversion for a certain period, and it makes a smooth transition.

Although there has been some work on the heterogeneous DDS [1, 2, 12, 13, 15–17], this area is relatively new and there is no proven solution for many technical problems.

DATAPLEX

DATAPLEX is a heterogeneous distributed database management system (HDDBMS) being developed by General Motors Research Laboratories. The key concept of its approach is to use the relational model as a common data model and the Structured Query Language (SQL) as a standard query language. This allows us to take advantage of the merits and wide acceptance of SQL and the relational model and to use results from the research on the homogeneous relational DDS.

The architecture of DATAPLEX is an open architecture which provides a well-defined interface that can

be extended to any DBMS and file system. A prototype system has been developed that interfaces a relational DBMS and a non-relational DBMS using the DATAPLEX approach. The prototype system processes all the test transactions correctly, and the performance of the system is reasonable. Based on the success of the prototype system, General Motors Research Laboratories have initiated the development of a full-function DATAPLEX. This article presents the architectural design of DATAPLEX and a prototype system that validates a major portion of its architecture.

ARCHITECTURE

The architecture of DATAPLEX consists of fourteen functionally-independent modules. This architecture is based on the relational model of data. Different data models used by unlike database systems structure data differently. The data definition used by each database system is called the local schema. The data definition of all the sharable databases in the heterogeneous DDS is transformed to an equivalent relational data definition. This common relational data definition is called the conceptual schema. The conceptual schema will include the relational data definition of only the stored data objects and local views. Each user's view of data in the heterogeneous DDS is called the external schema which consists of a subset of the conceptual schema and the global views derived from the data objects in the conceptual schema.

The relational model has been selected for the conceptual schema and the external schema for the following reasons:

- The relational model is easy for users to understand, and it supports high-level query languages.
- A relational transaction can be translated to a program in a low-level non-relational data manipulation language (DML) whereas the reverse translation is very difficult.
- The derivation of global views and the translation between the external schema and the conceptual schema are simple. These can be done in the same way as existing relational DBMSs do them.
- Relational DBMSs are wide-spread. For relational DBMSs, the translation between the conceptual schema and the local schema is trivial.

Since we use a common data model for the external schema, a uniform user interface can be provided. Among several relational query languages, we have chosen SQL as the uniform user interface because SQL is widely used and there is a movement to make SQL a standard relational query language.

Transactions will be translated at the location of the computer in which the data referenced by the transaction is stored (data location) rather than the location of the computer from which the transaction is originated (user location). The translation at the data location makes it possible to replicate only the crucial information such as the conceptual schema and the location of data in the DATAPLEX dictionary at every location. The detailed information necessary for transaction translation and query optimization can be stored at the data location. A query involving data at multiple locations is decomposed at the user-location into subqueries each of which references data from a single location.

The above strategies establish the architecture of DATAPLEX. Figure 1 shows DATAPLEX and other elements in a heterogeneous DDS. The names and functions of the fourteen modules that constitute the architecture are as follows:

- *Controller*—sequences and invokes necessary modules to process a transaction depending on the type of the transaction. This module supplies necessary parameter values to each of the other modules and processes multiple transactions at the same time by means of multitasking.
- *User Interface (basic user interface)*—prompts a user to issue SQL queries in a line or full-screen mode and returns messages and/or results to the terminal.
- *Application Interface*—sets up communication between an application and DATAPLEX. This module identifies SQL statements in the application and records an SQL transaction in the transaction log as an attempted transaction. It returns messages and/or data to the application and updates the status of a completed transaction as a successful transaction.
- *SQL Parser*—scans and parses SQL statements to check syntactic errors. It checks whether a query references correct names of relations and attributes. It transforms a query that references global views to a query that references only stored data objects and local views.
- *Data Dictionary Manager*—provides facilities to create a DATAPLEX dictionary. It provides facilities to query or update information in the DATAPLEX dictionary including the location of referenced data.
- *Security Manager*—enforces the content-independent access control using the global data object names and the userid. It enforces the content-dependent access control using the views as protection objects.
- *Distributed Query Decomposer*—decomposes a distributed retrieval query into local queries and user-location queries which merge local results. (A local query references data from a single location which may be a remote location.) It decomposes a distributed update query into a distributed retrieval part

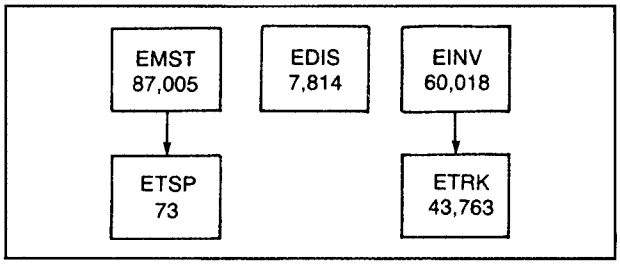

FIGURE 1. DATAPLEX and Other Elements in a Heterogeneous DDS

and a distributed update part. It decomposes the distributed retrieval part as above and generates a set of local update queries (one for each relation to be updated).

- *Distributed Database (DDB) Protocol*—interfaces the underlying communication protocol (or medium). This module should be easily adaptable to different communication protocols. It exchanges commands and data with the remote DDB Protocol using the file and message transfer facilities of the underlying communication protocol. It provides necessary functions of these facilities that are not provided by the communication protocol. It detects and reports any problem associated with the accessing of remote data.
- *Translator*—translates an SQL transaction to another relational transaction or a non-relational database program using a syntax transformation table from SQL to another relational query language, a set of rules to transform a non-relational data definition to an equivalent relational data definition, and stored information that can be lost during the transformation of a non-relational data definition to a relational data definition. It translates between global names and local names when they are different. It performs any necessary conversion of field formats such as the data type and the field length.
- *Local DBMS Interface*—interfaces a local DBMS. It passes the translated transaction and obtains the result.
- *Distributed Query Optimizer*—collects data-volume information and statistical information of local results from the locations involved in a distributed query and schedules an optimal data reduction plan. It executes the data-reduction plan by sending commands to the involved locations.
- *Distributed Transaction Coordinator*—requests both locks and lock releases on local data in a manner to correctly process a distributed transaction. It detects and handles global deadlocks. It enforces update atomicity in processing a distributed update and an update for multiple copies of data files in case of local system or communication network failures.
- *Relational Operation Processor*—is capable of processing relational operations for merging local results, reducing data during distributed query optimization, and further manipulating intermediate results locally.
- *Error Handler*—corrects recoverable errors, aborts the transaction, and cleans up the system for a non-recoverable error. It sends an appropriate message to a user and records it in the transaction log.

The way of packaging the above modules into processes depends on the detailed design consideration. In this section, we assume that DATAPLEX is one process for simplicity. DATAPLEXs at the user location and the data location(s) communicate through the DDB Protocol. The modules Translator and Local DBMS Interface are not used at the user location unless the user location is the same as the data location. Similarly, the modules User Interface, Application Interface, SQL Parser, and Distributed Query Optimizer are not used at the data location.

This architecture is independent of the local data system except for the modules Translator and Local DBMS Interface. Any data system can be interfaced to DATAPLEX by developing these two modules for it. Also, different communication protocols can be used by adapting the DDB Protocol to them. The solutions to technical problems, developed for some of the modules, are described next.

Distributed Retrieval

Our query decomposition generates local queries in SQL in a textual form rather than other equivalent representations used by some relational DDBMSs. This is an important consideration in a heterogeneous DDS because the translation is basically the text manipulation. Also, some of the major non-relational DBMSs will provide an SQL interface soon. SQL allows nesting of a query within another query. A nested query can be processed as a sequence of non-nested queries by processing the inner queries in the nest first. Also, the nesting can be eliminated by transforming a nested query to an equivalent non-nested query [11]. A non-nested query is transformed into a set of aggregate-free conjunctive queries. The part of the non-nested query requiring aggregate operations (e.g., set functions, `GROUP BY`, `ORDER BY`) forms a separate query which performs aggregate operations on the result of the remaining aggregate-free query.

A non-conjunctive query is decomposed into a set of conjunctive queries containing only `AND` Boolean operators:

(1) `NOT`'s are eliminated by using DeMorgan's Law and negating relational operators such as equals to (=) and greater than (>).
(2) `OR`'s and parentheses are eliminated by transforming the predicate into a disjunctive normal form and processing each conjunctive term as a separate query.

The result of the non-conjunctive query is the union of the results produced by the conjunctive queries. Each distributed conjunctive query is transformed to a query graph with relations as nodes and qualification terms as edges. Using the location information, the edges that correspond to global join terms are deleted from the query graph, and the global join terms are assigned to the qualification of the user-location query. Consequently, the query graph is decomposed into connected sub-graphs that are transformed back to local queries. The algorithm that decomposes a distributed conjunctive query into a set of local queries and a user-location query and the proof of the correctness of the algorithm are shown in [4].

If there are multiple data systems in a computer, a local query to this computer has to be further decomposed into a set of queries. Each must reference data from a data system. In this case, the location becomes the system identification of a data system. Our query-

decomposition method can also be used for this case without any modification. Our data-reduction plan for distributed query optimization [5] is a sequence of semi-joins which consists of local data-reduction operations and data moves among computers. The plan generated by the distributed query optimizer significantly reduces the amount of data moved among computers in processing a distributed query.

Distributed Data Security

The data security enforcement of DATAPLEX is a content-dependent access control (CDAC). In a CDAC, access rules can include a predicate whose truth depends on the content of data; whereas, in a content-independent access control (CIAC), access rules are defined only on the types of data. Since not all DBMSs support CDAC it must be handled by DATAPLEX. A CDAC in a heterogeneous DDS is described in [16, 17]. Our access control is similar to the above one in that views are used as protection objects [3]; however, it differs from the above one in that DATAPLEX does not materialize views as temporary relations. In a DATAPLEX environment, the security is enforced in two steps: 1) first by DATAPLEX and 2) then by local DBMSs. The access control by DBMSs preserves the local autonomy.

A global database administrator (GDBA) creates a DATAPLEX dictionary and grants to users access privileges to global data objects in the dictionary. The GDBA may create a view that contains a predicate to enforce a CDAC. When the GDBA grants access to a global data object to a user, the following happens:

(1) If the global data object corresponds to a local data object managed by a local DBMS (this may be a local view if the DBMS supports views), the user becomes an authorized user of the local data object at the location of the data object.
(2) If the global data object is a global view defined on one or more local data objects (these may have been created by different users possibly at different locations), DATAPLEX becomes an authorized user of the local data objects at the location of the data objects.

When a user issues a query to DATAPLEX, the user-location DATAPLEX performs a content-independent security check using the userid and the global data objects referenced in the query. The processing of the query continues only if the check succeeds. If the global data objects in the query are all local data objects, the query and the userid are sent to the location of the data objects where a local DBMS checks the security. In case of a distributed query, the query is decomposed and subqueries are sent along with a userid.

If the query references a global view, the query is modified [14] using the definition of the view. The resulting query references only local data objects and contains the predicate, if any, in the view definition that enforces a CDAC. The modified query is sent to the location of the local data objects along with DATAPLEX (i.e., security code of DATAPLEX) as a userid rather than the original userid. Since DATAPLEX is authorized to access the local data objects as described above in (2), the modified query passes the local security check. In this method DATAPLEX becomes a *super user* to a local DBMS. This is similar to the situation in which a DBMS becomes a *super user* to a file system so that a user's access, checked by the DBMS, will not be checked again using the userid by the access control of the underlying file system. The user's request becomes an access by the DBMS to files.

Distributed Update

There are three problems associated with the distributed update: distributed concurrency control, distributed deadlock handling, and distributed data recovery. Since these problems require more research, we only outline our approach. Local DBMSs provide concurrency control, deadlock handling, and recovery for local databases. We assume that local concurrency control is based on the locking, which is true for most, if not all, commercially available DBMSs. Our method for distributed concurrency control is to globally perform the two-phase locking [9]. Since local DBMSs perform the two-phase locking locally, an additional task of DATAPLEX is to prevent local DBMSs from releasing locks until update processing at other involved locations is complete.

DATAPLEX will initially use the time-out method to detect global deadlocks. In this method, no response from a target DATAPLEX within a pre-arranged time is regarded as a global deadlock. Therefore, the method is based on a necessary condition for the deadlock. The advantage of the time-out method, however, is that it is simple and applicable to both the homogeneous DDS and the heterogeneous DDS.

A basic method for recovery is the two-phase commit to enforce the update atomicity. The two-phase commit consists of a prepare-to-commit (PTC) followed by an actual commit. Once a DBMS reaches a PTC state, the recovery to this state or to the initial state prior to an update processing is possible. The distributed two-phase commit can be done if local DBMSs provide the PTC state at the request of DATAPLEX. DBMSs currently do not provide an external interface to perform PTC. They internally perform a PTC followed by a commit in response to an external commit request. Therefore, this is an open problem that requires more research, and it is an important area for standardization of DBMSs to provide an external interface for PTC. Since the distributed two-phase commit forces local DBMSs to commit and release locks only after all DBMSs involved in a distributed update are prepared to commit, this automatically satisfies the distributed concurrency control by enforcing global two-phase locking.

TRANSLATION

In a heterogeneous DDS, different local DBMSs use different data models and provide different DMLs. The

translation of data definition and DML is necessary to provide the relational model as a common data model and SQL as a uniform user interface.

Non-relational data models that require the data definition translation are the hierarchical model and the network model. The major difference between the relational data model and the non-relational data model is the way of modeling relationships among entities. The relational model represents the relationship between two entities using a relation (data table); whereas, the network model and the hierarchical model represent the relationship using an access path (pointer).

Two related entities have either a one-to-many relationship or a many-to-many relationship. These relationships are modeled in the hierarchical or network model as below. The data structure diagram of the division record and the plant record and their one-to-many relationship is shown in part (a) of Figure 2. An example of the many-to-many relationship is the relationship between the frame and the part. The many-to-many relationship is modeled by identifying two one-to-many relationships using an intersection data. The data structure diagram representing the many-to-many relationship between the frame and the part is shown in part (b) of Figure 2.

Our basic method for translating a non-relational data definition to an equivalent relational data definition is the use of the key in the owner record as a foreign key in its member records. This is intended to produce a simple and normalized set of relations. The data definition depicted in Figure 2 is translated as follows:

One-to-many relationship

```
DIVISION   (DIV#, DNAME)
PLANT      (DIV#, PLT#, STATE)
```

Many-to-many relationship

```
FRAME      (FR#, FNAME)
ASSEMBLY   (FR#, P#, QTY)
PART       (P#, MTRL)
```

The navigation through access paths in non-relational databases can be accomplished by equi-joining relations in an equivalent relational data definition. For example, the access to part data used in certain frames is the same as joins by FRAME.FR# = ASSEMBLY.FR# and ASSEMBLY.P# = PART.P#.

A repeating group in a non-relational data definition can be translated to an additional relation or to sets of attributes (the number of the sets is the maximum number of repetitions). Therefore, an equivalent relational data definition is always in first normal form, a prerequisite for using a relational query language. Relations in higher normal forms, however, cannot be generated unless the existing non-relational data definition is normalized. Non-relational databases also have anomalies without normalization. This situation is illustrated in the following example:

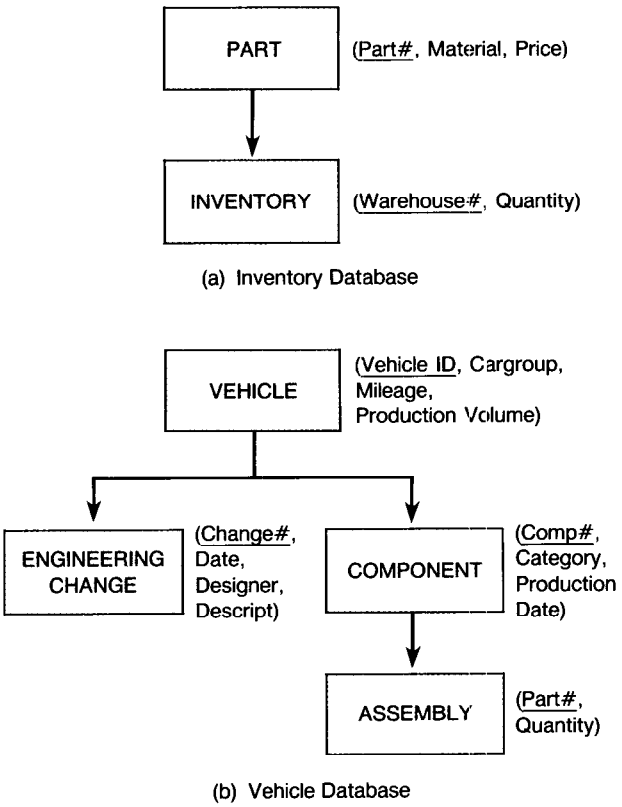

FIGURE 2. The Data Structure Diagram of the Entity Relationship

Consider an IMS database with one segment SUPPLY (S#, CITY, P#, QUANTITY) where the key consists of S# and P#. If a record is inserted with only the values of S# and CITY specified, the values of P# and QUANTITY are unpredictable. Therefore, the data for a supplier in a city cannot be inserted until the supplier supplies a part. An equivalent relational data definition of SUPPLY, which is the same as the IMS segment definition, is not in second normal form because CITY functionally depends on S# which is a partial key. The correct design of the IMS database consists of two segments:

A parent segment: SUPPLIER (S#, CITY) with S# as a key
A child segment: SUPPLY (P#, QUANTITY) with P# as a key

This design does not have the above insertion anomaly. An equivalent relational data definition of the two segments consists of two relations SUPPLIER (S#, CITY) and SUPPLY (S#, P#, QUANTITY). The key of SUPPLIER is S# and the key of SUPPLY consists of S# and P#. These relations are in fifth normal form because all the functional dependency is on the keys, and

there is neither multi-valued dependency nor join dependency.

Once the conceptual schema is set up in relational data definitions, external schemas are created from the conceptual schema, and users can formulate SQL transactions. DATAPLEX must translate an SQL transaction that references relations in the conceptual schema to a transaction in another relational DML or a non-relational DML. Since the translation among different relational DMLs is straightforward, we will only consider the translation to non-relational DMLs.

The relational data model and the non-relational data model represent the same information differently. The DML used to formulate requests on information is tightly coupled with the particular representation of the information. Therefore, two basic components for generating a transaction in a particular DML are the semantics of the information request (what information is requested?) and the data definition which results from data modeling.

In order to translate an SQL transaction to a non-relational database program, the semantics of the information request and the non-relational data definition of the data referenced by the transaction must be obtained. The semantics of the information request can be obtained from the SQL transaction. In fact, SQL is specifically designed to represent the semantics of an information request. The semantics include relation names, attribute names, and conditions on attribute values. In addition, access paths in the non-relational data definition can be partially derived from the SQL transaction. This can be achieved by applying an inverse of the data definition translation to the join terms of the SQL transaction because the access path in the non-relational data definition has been translated to joining attributes in the relational data definition.

Nonetheless, some information about the non-relational data definition is lost in the process of data definition translation. For example, the join term identifies the existence of an access path between the two records but cannot determine which one is the owner record. Also, an attribute corresponding to a foreign key in a relation may not exist in the corresponding record. This misplaced attribute can also hide an actual access path. Therefore, the information that can be lost in data definition translation must be stored in the translation table and used in the transaction translation. This discussion is illustrated in Figure 3.

It is possible that the name of a record or a relation in a local database is used in another local database. Each relation in the conceptual schema must be distinctly named. Therefore, the names of relations in the conceptual schema can be different from those of the corresponding records or relations in the local schema. In this case, the names in the conceptual schema have to be translated to the corresponding local names in the local schema. When the result is sent to the user-location, the reverse translation is necessary. In addition, the format and/or name of a field may need to be translated when the format and/or name of the same field is defined differently at different locations.

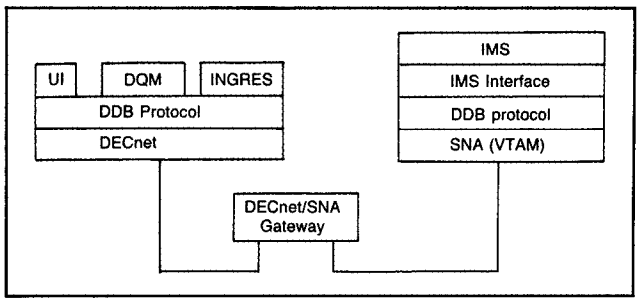

FIGURE 3. The Relationship between Data Definition Translation and Transaction Translation

The location of the transaction translation is important from the performance and system-maintenance standpoint. We select the data location to translate transactions for the following reasons:

- The translation at the user location would require each DATAPLEX to store translators and the translation tables for shared databases at all locations. Any change of the translation table to incorporate local data definition change would have to be propagated to all the locations.
- The size of a non-relational database program is generally much larger than that of an equivalent SQL transaction. Therefore, the user-location translation incurs a higher communication cost.
- The data-location translation increases parallel processing for distributed transactions.

PROTOTYPE SYSTEM

The prototype DATAPLEX interfaces IMS hierarchical DBMS on IBM/MVS and INGRES relational DBMS on DEC/VMS. The prototype system allows users and applications to retrieve data from IMS and/or INGRES with a single SQL query from DEC/VMS such that the location of data is transparent to requestors. The unique features of the prototype system are as follows:

(1) SQL queries to IMS
(2) Distributed SQL queries to IMS and INGRES
(3) Distributed SQL queries embedded in a C language program

The following data formats are supported for IMS and INGRES databases:

- Characters
- Text (variable length) fields with maximum 2000 bytes
- Integers in 2 bytes or 4 bytes
- Floating point numbers in 4 bytes or 8 bytes
- Packed decimal numbers, supported only by IMS

Since rapid prototyping was required to show the feasibility before developing a full-function DATAPLEX,

the update of IMS data and the full query optimization were not implemented in the prototype system. In addition, the system supports a subset of SQL defined to have the following syntax:

SELECT	list of target attributes and set functions
FROM	list of relations
WHERE	qualification
ORDER BY	attributes

where set functions are MAX, MIN, SUM, COUNT, AVERAGE, and the qualification contains >, >=, <, <=, =, <>, AND, OR, NOT, and parentheses.

Development

The core of the prototype system was structured in four processes as follows:

(1) *User Interface (UI)*—this process corresponds to the DATAPLEX modules: User Interface and Application Interface.

(2) *Distributed Query Manager (DQM)*—this process performs four major functions:

- Distributed data dictionary management
- Query parsing
- Decomposition and generation of an execution plan of distributed queries
- Execution of query plans through communication with local DBMSs

This process corresponds to the DATAPLEX modules: SQL Parser, Data Dictionary Manager, Distributed Query Decomposer, and a part of Distributed Query Optimizer.

(3) *IMS Interface*—this performs the following functions:

- Receives queries from the Distributed Query Manager at the user location and sends back results
- Translates SQL queries to DL/1 programs processable by IMS
- Local interface to IMS to submit DL/1 programs and return IMS data

This process corresponds to the DATAPLEX modules: Translator and Local DBMS Interface.

(4) *DDB Protocol*—program-to-program communication is required to support the logical interface between the IMS Interface on IBM/MVS and the DQM on DEC/VMS. A DECnet/SNA GATEWAY [7] from DEC was used to connect IBM's Systems Network Architecture (SNA) network and DEC's DECnet. The primary functions provided by the DDB Protocol are as follows:

- Establishment of a session between DQM and IMS Interface
- A file transfer facility between DQM and IMS Interface to exchange SQL queries and results
- Data format conversion between IBM 370 and DEC VAX
- Interface to the software used by the DECnet/SNA GATEWAY

The overall architecture of the prototype system is shown in Figure 4. On a DEC computer, DDB Protocol provides communication among the processes of User Interface, DQM, and INGRES. DDB Protocol also provides the interface via DECnet and the DECnet/SNA GATEWAY to the DDB Protocol running on the IBM side. Some modules on IBM/MVS were implemented in IBM assembler. The C language was used to implement all other modules. The data security is locally enforced. The prototype system sends a query to a DBMS with a userid. INGRES enforces a CDAC; whereas, the IMS Interface enforces CIAC by checking whether a user is authorized to access IMS data at a segment level using the userid.

Test Bed

A test bed has been established at General Motors Research Laboratories. IMS is installed on an IBM 4381 MODEL 14. The IBM 4381 is a departmental test machine which runs three MVS/XAs on top of VM. INGRES is installed on a DEC VAX 11/785. Users run the prototype system through the VAX/VMS. A test distributed database and test transactions were created. The test distributed database consists of two IMS databases and an INGRES database. The two IMS databases are named VEHICLE database and INVENTORY database. The database definitions and field definitions for application programs for the two IMS databases are represented in the data-structure diagram in Figure 5. The underlined fields are the key fields of the segments.

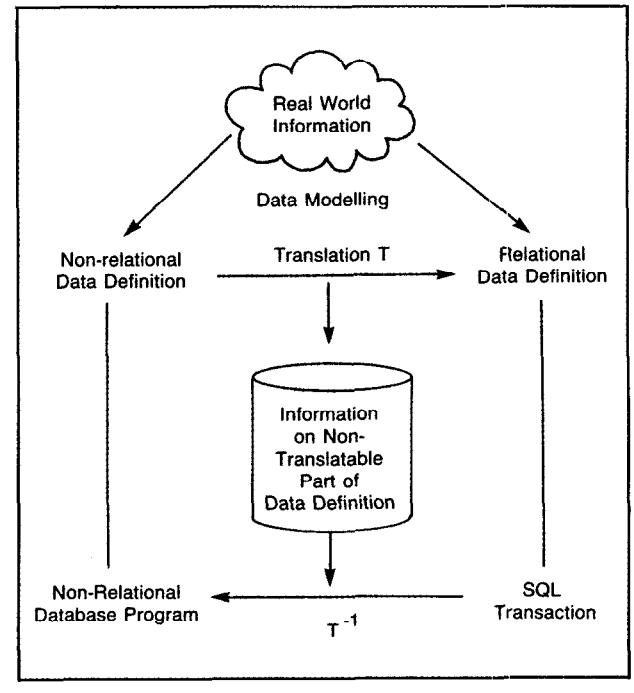

FIGURE 4. The Architecture of DATAPLEX Prototype

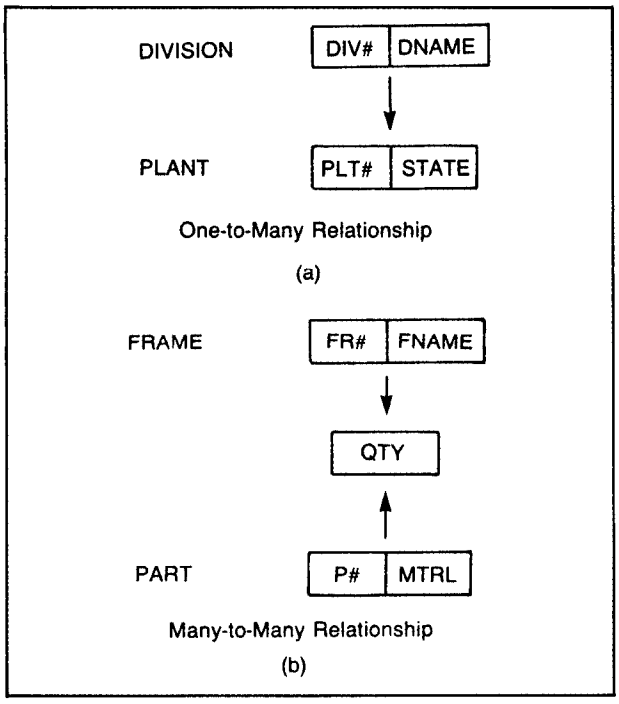

FIGURE 5. Test IMS Databases

An equivalent relational data definition of the IMS databases is given below. The data type and format of each attribute is shown in the parenthesis following the attribute name. C denotes a character data type, I an integer, F a floating point number, and TEXT a variable-length character string. The length of a field is specified in number of bytes following the data type.

```
VEHICLE (VID(C3), CARGROUP(C3), MILEAGE(F8),
  PRODVOL(I4))
ENGCHA (VID(C3), CHGNUM(C5), CHGDATE(C8),
  DESIGNER(C15), DESCRIPT(TEXT MAX 50))
COMP (VID(C3), CNUM(C3), CATEGORY(C10),
  PRODDATE(C8))
ASSEMBLY (VID(C3), CNUM(C3), PNUM(C3),
  QTY(I2))
PART (PNUM(C3), MTRL(C10), PRICE(F8))
INVENTRY (PNUM(C3), WHNUM(C3), QOH(I4))
```

This definition is derived using the method explained earlier. The INGRES database consists of the following two relations:

```
  SUPPLIER (S_NUM(C3), LOC(C10),
STATUS(C1))
  SUPPLY (S_NUM(C3), P_NUM(C3), QTY(I4))
```

Users formulate requests based on the relational view of the distributed database which consists of the above eight relations. The location and the type of the actual database are transparent to users. The sizes of the databases are small so that the correctness of the results of test queries can be checked easily. The number of rows in each relation is between five and fifty.

Results

Twenty test transactions are formulated to check various functions of the prototype system: fourteen SQL queries to IMS databases, five distributed SQL queries, and an embedded distributed SQL query in a C program. Representative queries are listed in Appendix A. The prototype system processes all the test transactions correctly.

The efficiency of the prototype system is measured and analyzed in terms of query processing time. The time is the clock time because the CPU time does not include the data transmission time in a communication network. The average query response time is about 37 seconds when the prototype system runs on an IBM 4381 and a DEC VAX 11/785. If an IBM 4381 is replaced by an IBM 3090, the average response time is estimated to be 12 seconds. It is observed that the IMS Interface is the bottleneck of the prototype system for processing queries against small or medium-sized databases.

Production IMS data is used to test the effect of the size of the database on efficiency. The data is from the Maintenance Management Information System (MMIS) running at a car assembly plant. The size of the MMIS database is about 1,000 times larger than that of the test IMS database. Three SQL queries are executed against a part of the MMIS database whose data structure diagram is shown in Figure 6. An equivalent relational data definition of the MMIS database in Figure 6 and the SQL queries are listed in Appendix B.

The requests formulated as the three SQL queries were also programmed in PL/I, and the PL/I programs are executed locally on the IBM 4381 against the MMIS database. The processing time of the SQL queries by the prototype system and by the corresponding PL/I programs is tabulated in Table I. Since the clock time varies considerably for different executions, Table I provides the average time for five executions of the test queries and programs. The result in Table I shows that the prototype system incurs overhead compared with the access using PL/I programs. As the database size grows, the fraction of the overhead to the total processing time decreases.

A TSO interface to the IMS Interface is developed to issue SQL queries directly from a TSO terminal. Since all processing for such queries is performed on an IBM 4381, the CPU time can be measured. Three sample requests for retrieving the MMIS data are obtained from plant data-processing personnel. These requests are formulated in SQL queries and PL/I programs, and they are executed three times locally on the IBM 4381. In contrast to the variance of the clock time, the variance of the CPU time is very small. The SQL queries are listed in Appendix B, and the average execution time is given in Table II.

The execution of Request 6 is very fast because the data is searched using an index on einvkey attribute. The test result shows that the performance of the IMS Interface is reasonable, and its overhead consists of a

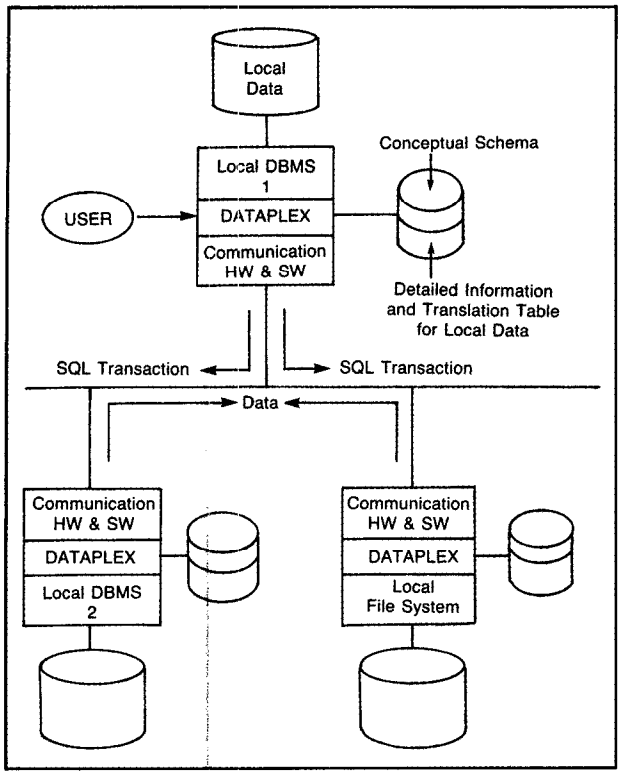

FIGURE 6. A Part of the MMIS Database with the Number of Occurrences

TABLE I. The Comparison of Retrieval Time of MMIS Data Using SQL Queries on Prototype System Versus PL/I Programs on IBM 4381

	Processing Time (seconds)		
Request	SQL Query	PL/I Program	Referenced segments
1	368	306	EMST
2	61	14	EDIS
3	346	260	EMST, ETSP

heterogeneous distributed database system provides a smooth migration path in the current environment where the technology is rapidly changing.

The architecture of DATAPLEX consists of fourteen functionally-independent modules. This architecture is modular and is an open architecture that provides facilities for easy interfaces to additional data systems. Functionality and performance can be increased with this architecture.

A DATAPLEX prototype has been developed that interfaces an IMS hierarchical DBMS on IBM/MVS and an INGRES relational DBMS on DEC/VMS. The capability of the prototype system shows the feasibility of the concepts underlying the DATAPLEX approach to solving the problem of transparently sharing data in a diverse database environment. The performance of the prototype system indicates that DATAPLEX can be used in a real data-management environment.

TABLE II. The Comparison of Retrieval Time of MMIS Data Using SQL Queries Versus PL/I Programs on IBM 4381

	Clock Time (seconds)		CPU Time (seconds)			
Request	SQL Query	PL/I Prog.	SQL Query	PL/I Prog.	Referenced segments	Number of rows in result
4	189.24	150.31	49.28	39.63	EINV	5
5	188.13	161.44	50.77	39.61	EINV	539
6	34.42	1.55	4.52	0.41	EINV, ETRK	165

fixed overhead and the overhead proportional to the amount of disk I/O. This overhead can be further reduced by developing an efficient IMS Interface.

SUMMARY AND CONCLUSIONS

Heterogeneous distributed database systems have advantages over centralized database systems and homogeneous distributed database systems for a large organization with diverse requirements. In addition, the flexible integration of diverse databases through the

Acknowledgments. I would like to thank George Dodd and John Joyce for supporting this project. The prototype system was jointly developed with Relational Technology, Inc. I appreciate Kenneth McCloskey's participation in the implementation of the prototype system. I also wish to thank the three anonymous reviewers for many valuable comments.

APPENDIX A
REPRESENTATIVE TEST QUERIES

(1) SQL queries to IMS

```
select  e.vid, descript, cnum,
        proddate
from    engcha e, comp c
where   chgdate > '85-12-31'
and     e.vid = c.vid
and     category = 'ELECTRICAL'
```

(2) Distributed SQL queries

```
select  s.s_num, p_num, a.vid, qty,
        prodvol
from    supplier s, supply y, part p,
        assembly a, vehicle v
where   s.s_num = y.s_num
```

```
(1) select  p.pnum, price, qty
    from    part p, assembly a
    where   p.pnum = a.pnum
    and     vid = 'V20'
    and     qty > 10

    select  avg(price)
    from    part
    where   mtrl = 'ALUMINUM'

    select  distinct *
    from    vehicle
    order by mileage desc

    select  pnum, price
    from    part
    where   price > 100
    or      price < 10
```

```
(2) and     y.p_num = p.pnum
    and     p.pnum = a.pnum
    and     a.vid = v.vid
    and     loc = 'Michigan'

    select  s.s_num, p_num, price
    from    supplier s, supply y, part p
    where   s.s_num = y.s_num
    and     y.p_num = p.pnum
    and     (status = 'A'
    or      price > 100)

    select  s_num, p_num, mtrl, qty
    from    supply y, part p
    where   y.p_num = p.pnum
    and     not mtrl = 'STEEL'
    and     qty > 100000
```

APPENDIX B
RELATIONAL DATA DEFINITION AND SQL QUERIES FOR MMIS DATABASE

(1) An equivalent relational data definition

Since relations contain a large number of attributes, only the attributes referenced in queries are listed for each relation.

```
EMST    (EMSTKEY, SET_SEQ, TASK_CD, SKILL_CD, EST_TIME)
ETSP    (EMSTKEY, COMMON_CD, QUANTITY)
EDIS    (ENTRY_DATE, ENTRY_TIME, DESC_DESC, SYSTEM, EQUIP)
EINV    (EINVKEY, COMP, DESCRIPTOR, EQUIP, PARTS_SET, PLANTSI1, PROPERTY, SYSTEM)
ETRK    (EINVKEY, DISP, BUILD_COST, DESIGN_COST, REJECT_CODE)
```

(2) SQL queries to MMIS database

```
1.  select  emstkey, set_seq, task_cd, skill_cd, est_time
    from    emst
    where   est_time > 10

2.  select  entry_date, entry_time, desc_desc, system, equip
    from    edis
    where   entry_date > '870201'

3.  select  m.emstkey, set_seq, task_cd, common_cd, quantity
    from    emst m, etsp t
    where   est_time > 10
    and     m.emstkey = t.emstkey

4.  select  plantsi1, system, equip, comp
    from    einv
    where   parts_set <> '    '
    and     property = 'REPAIR'

5.  select  plantsi1, equip, comp, descriptor, property
    from    einv
    where   system = 'SCRAP'

6.  select  disp, design_cost, build_cost, reject_code
    from    etrk
    where   einvkey <= 'TO 0000300'
```


REFERENCES

1. Breithart, Y. J., and Hartweg, L. R. RIM as an implementation tool for a distributed heterogeneous database. In *Proceedings of IPAD II Symposium on Advances in Distributed Database Management for CAD/CAM* (Denver, Co., Apr. 17–19). NASA, Hampton, Va., 1984, pp. 155–164.
2. Cardenas, A. F., and Pirahesh, M. H. Database communication in a heterogeneous database management system network. *Inf. Syst. 5*, 1980, 55–79.
3. Chamberlin, D. D. et al. Views, authorization, and locking in a relational database system. In *Proceedings of the National Computer Conference* (Anaheim, Calif., May 19–25). AFIPS, Montvale, N. J., 1975, pp. 425–430.
4. Chung, C. W. Design and implementation of a heterogeneous distributed database management system. In *Proceedings of the IEEE INFOCOM '89 8th Annual Joint Conference of the IEEE Computer and Communications Societies* (Ottawa, Ont., Canada, Apr. 25–27). IEEE, New York, 1989, pp. 356–362.
5. Chung, C. W., and Irani, K. B. An optimization of queries in distributed database systems. *J. Parallel and Distrib. Comput.* 3, 2 (June

1986), 137–157.
6. Dayal, U., and Goodman, N. Query optimization for CODASYL database systems. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM/SIGMOD, New York, 1982, pp. 138–150.
7. Digital Equipment Corp. *Networks and Communications Buyers Guide*, Maynard, Mass., 1986.
8. Elmagarmid, A. K. A survey of distributed deadlock detection algorithms. *ACM SIGMOD Record 15*, 3 (Sept. 1986), 37–45.
9. Eswaran, K. P. et al. The notions of consistency and predicate locks in a database system. *Commun. ACM. 19*, 11 (Nov. 1976), 624–633.
10. Gligor, V. D., and Luckenbaugh, G. L. Interconnecting heterogeneous database management systems. *Comput.* (Jan. 1984), 33–43.
11. Kim, W. On optimizing an SQL-like nested query. *ACM Trans. Database Syst. 7*, 3 (Sept. 1982), 443–469.
12. Landers, T., and Rosenberg, R. L. An overview of multibase. In *Distributed Data Bases*. North-Holland, Amsterdam, 1982, pp. 153–184.
13. Rahimi, S. K. et al. A structural view of Honeywell's distributed database testbed system: DDTS. *Database Eng. 5*, 4 (Dec. 1982), 283–287.
14. Stonebraker, M. Implementation of integrity constraints and views by query modification. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (Anaheim, Calif., May 19–23). ACM/SIGMOD, New York, 1975, pp. 65–78.
15. Su, S. Y. W. et al. The architecture and prototype implementation of an Integrated Manufacturing Database Administration System. In Spring COMPCON, 1986.
16. Wang, C. Y., and Spooner, D. L. Access control in a heterogeneous distributed database management system. Tech. Rep. 86-1, Rensselaer Polytechnic Inst., 1986.
17. Wang, C. Y. Discretionary access control in a heterogeneous distributed database management system, Ph.D. Thesis, Rensselaer Polytechnic Inst., 1986.
18. Zaniolo, C. Design of relational views over network schemas. In *Proceedings of ACM SIGMOD International Conference on Managenemt of Data* (Boston, Mass., May 30–June 1). ACM/SIGMOD, 1979, pp. 179–190.

CR Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed databases*; H.2.3 [**Database Management**]: Languages—data manipulation languages (DML); *query languages*; H.2.5 [**Database Management**]: Heterogeneous Databases—*data translation; program translation*
General Terms: Design, Languages, Standardization
Additional Key Words and Phrases: Heterogeneous distributed database management system, prototype system

ABOUT THE AUTHOR:

CHIN-WAN CHUNG is currently a staff research scientist in the Computer Science Department at General Motors Research Laboratories. His research interests include distributed databases, object-oriented databases, and expert database systems. Author's Present Address: Computer Science Department, General Motors Research Laboratories, Warren, MI 48090-9057. CSNET: CHUNG@GMR.COM

CORRIGENDUM

In the Computing Practices article,"DATAPLEX: An Access to Heterogeneous Distributed Databases," by Chin-Wan Chung, published in the January 1990 issue of Communications, the order of the figures were erroneously reversed.

Below is the correct order of the figures. Please refer to this corrected version when reading the original article.

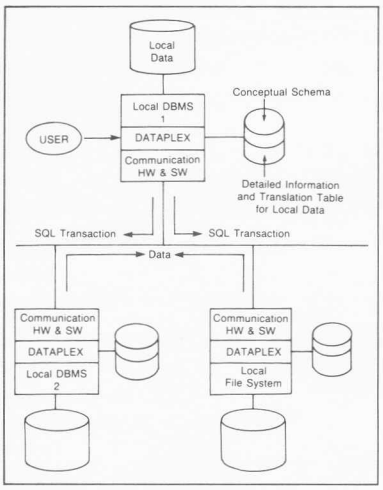

FIGURE 1. DATAPLEX and Other Elements in a Heterogeneous DDS

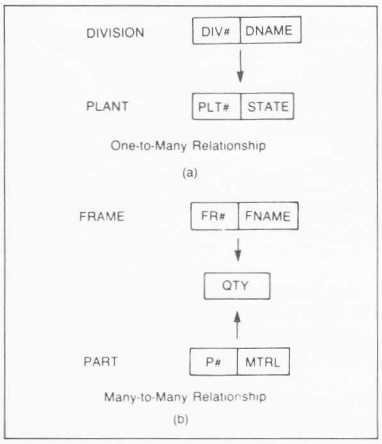

FIGURE 2. The Data Structure Diagram of the Entity Relationship

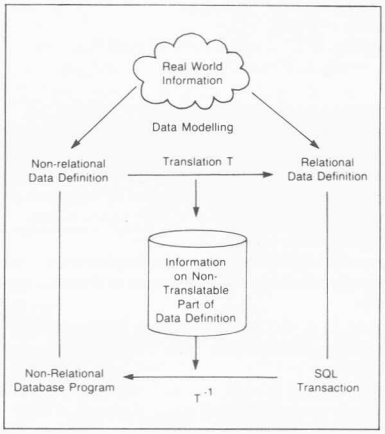

FIGURE 3. The Relationship between Data Definition Translation and Transaction Translation

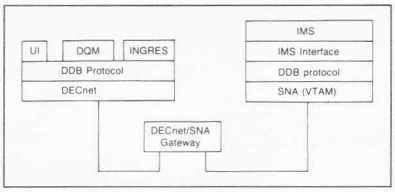

FIGURE 4. The Architecture of DATAPLEX Prototype

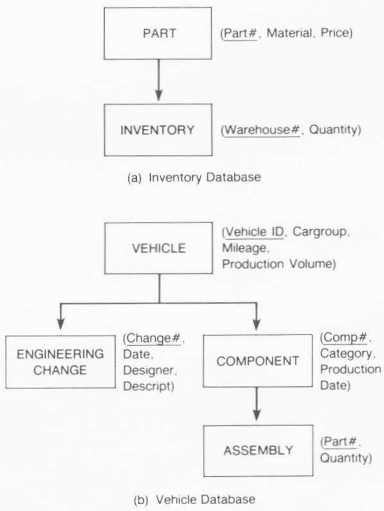

FIGURE 5. Test IMS Databases

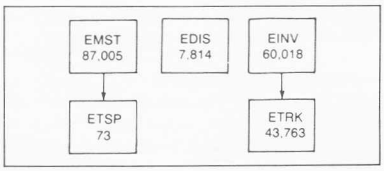

FIGURE 6. A Part of the MMIS Database with the Number of Occurrences