Myriad: Design and Implementation of a Federated Database Prototype

EE-PENG LIM, SAN-YIH HWANG, JAIDEEP SRIVASTAVA,* DAVE CLEMENTS AND M. GANESH

Department of Computer Science, University of Minnesota, 4-192 EE/CS Bldg, 200 Union St. SE, Minneapolis, MN 55455, U.S.A.

SUMMARY

A key problem in providing 'enterprise-wide' information is the integration of databases that have been independently developed. An important requirement is to accommodate heterogeneity and maintain the autonomy of component databases. Myriad is a federated database prototype developed at the University of Minnesota, to provide a testbed for investigating alternatives in architecture and algorithms for database integration, query processing and optimization, and concurrency control and recovery. The system incorporates our group's research results in these areas. This paper describes our experiences in the design and implementation of Myriad, and in the project management. Special emphasis is given to discussing design alternatives and their impact on Myriad. This paper also presents the software engineering principles and the project management techniques we used in developing Myriad and the lessons we learned. We believe these lessons would be useful for practitioners who wish to develop a similar system.

Handling heterogeneity and autonomy were prime objectives throughout the prototyping effort. We are convinced that a prototype federated database is an important infrastructural requirement for the overall goal of 'enterprise-integration', and believe Myriad to be a significant contribution towards this.

INTRODUCTION

The ability to access data that resides in multiple autonomous and heterogeneous datasources, in a uniform and integrated manner, is becoming very important to organizations, as the realization grows that information is a vital organizational asset. Apart from system-level incompatibilities between different database systems, incompatibilities also exist between the content of different but related databases. The heterogeneity and autonomy characteristics of the local database systems cause a variety of difficulties in both efficient processing of global queries and the correct execution of transactions which have to satisfy global serializability. Moreover, heterogeneity among the databases' content has to be shielded from the global users who want to view the collection of databases as an integrated entity.

Myriad is a federated database system (FDBS) prototype developed at the University of Minnesota to satisfy the above needs. It seeks to provide global transaction management and query processing over a set of autonomous and heterogeneous DBMSs on which pre-existing databases were designed and implemented. It acts as a testbed for validating

---

* All correspondence should be directed to Professor Jaideep Srivastava.

and comparing solutions to various FDBS problems such as transaction management and query optimization. It also provides an environment for realizing new federated database applications.

Related work

In the last decade a number of database integration projects have focused on schema integration[1,2] leading to some commercial products. However, few supported good query capabilities, and none supported transaction management. A number of FDBS projects are currently underway. The *Multibase*[3] project developed the concept of generalized hierarchies to integrate heterogeneous schemas. It focused on query processing, but did not address transaction management issues. In *Multidatabase*,[4] there is no integrated schema over local databases. The result of processing a global query is presented as a set of relations, each of which corresponds to the result of a local query decomposed from the global query. Multidatabase also does not provide transaction management. *Pegasus*[5] is an ongoing project at HP Labs which focuses on using an object-oriented data model to integrate local databases. While original plans did include transaction management, the actual approach taken by Pegasus has not been described in the literature. *Interbase*[6] provides a tool-based interface to execute global transactions without violating the autonomy of local DBMSs. To execute a global transaction, a user has to specify a set of subtransactions (one for each server site) together with their interrelationships, i.e. the decomposition of global transactions into sub-transactions is not automated by Interbase. Interbase does not provide any integrated view over existing databases. Hence, global query processing and optimization have not been considered. Thus, Myriad is the first system to provide schema integration, query processing and optimization, and transaction management capabilities in a single framework.

Myriad's strengths

In Myriad, a federation consists of an integrated database the schema of which is represented as sets of integrated relations. SQL, mainly due to its simplicity and popularity among database users and vendors, has been adopted to express global queries as well as the queries for the local database gateways. The Myriad prototype addresses global transaction management and query processing in an integrated manner. The main contributions of the Myriad prototype developement include:

1. designing a flexible FDBS architecture for accommodating and experimenting with different query processing strategies and transaction management algorithms;
2. providing a site transparent environment in which local databases can be presented in both a tightly and a loosely integrated manner to the global users;
3. defining and implementing a useful set of integration operations which include user-defined functions to resolve data incompatibities between independently designed local databases;
4. developing a query processing strategy in which local DBMS sites with different query processing capabilities can collaborate in a distributed manner to compute global query results.
5. supporting the ACID transaction model as the atomic unit of interaction at the global level while providing a general-purpose programming facility to co-ordinate sequential or concurrent execution of multiple global transactions.

Software development experience

We employed several software engineering and project management techniques in Myriad software development. These techniques include standard coding conventions, event-driven code mapping, standard debugging and testing procedures, code walk through, documentation, and project schedule monitoring. While these techniques in general were found to be useful in our software design, we discovered that in some cases the realization of these techniques could be further improved. For example, coding standard should not only be comprehensive in order to accomodate enough information but also be concise to give programmers flexibility in writing their code; including more information in a debugging message, such as site and process identifier, helps the programmers locate the problems; a flexible project schedule keeps the project development on schedule even in the presence of unanticipated events or difficulties.

In the following sections we describe in detail our database integration philosophy, the system architecture used to realize the integration, and the design and implementation of various system components. As Myriad required a substantial effort that involved several developers and produced about 35,000 lines of code, project management was a crucial aspect. We have also presented our experience in managing the Myriad project. Finally, we provide our conclusions and directions for future extension.

FEDERATED DATABASE INTEGRATION

A federated database system is one that provides site-transparent access to multiple component databases. The user of an FDBS system poses queries against a federated (global) schema or the export schemas of the component databases. These queries are translated into queries on the individual databases by the FDBS. In this section we describe our approach to integrating heterogeneous databases, construction of global schemas and the operations we use to achieve this.

Framework

The integration framework adopted by Myriad is shown in Figure 1. One or more existing local databases can participate in a federation in which an integrated database is defined. An integrated database comprises of a global schema which is a set of relations whose data are derived from the local databases. These derived relations are also known as integrated relations. In Myriad, each local database that participates in an integrated database must provide a relational export schema. A local database may have multiple export schemas for different integrated databases. An export schema can also be shared among different integrated databases. Each export schema contains a set of relations on which relational queries can be posed. Relational queries on an export schema are translated appropriately into local database queries which may be non-relational. The relational to local query translation is performed by the gateways residing on the local database systems.

While the translation between export schema and local database is handled by the gateway customized for the local DBMS and DB, the resolution of data incompatibilities between export schemas is an essential task in Myriad query processing. Incompatibilities exist because the local databases represented by these export schemas have been designed and developed independently, often by different teams of people, to meet different application requirements. In order to process federated queries over a global schema, the mapping from
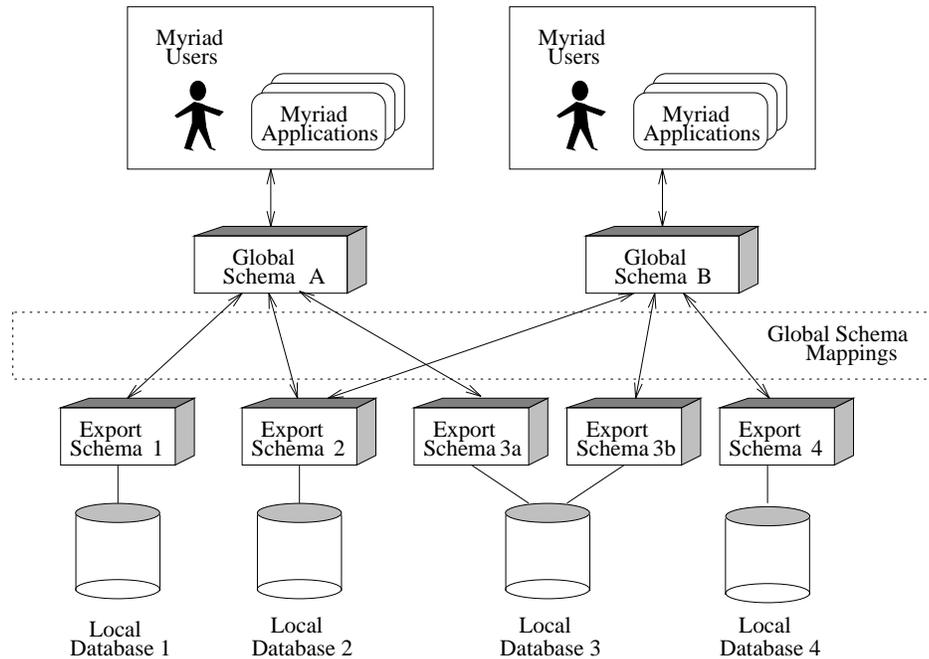
Figure 1. Myriad schema architecture.

the integrated relations to the export relations has to be defined. We call this the *global schema mapping* task. In the following subsection, we represent a global schema mapping as trees of integration operations which define the computation of integrated relations from the set of export relations.

Integrating local schemas

Myriad assumes each exported local schema is in the relational model.* The job of schema integration is to integrate several local relations, some of which may be located in different databases controlled by different DBMSs, to form a global relation. In Myriad, we have focused on two instance-level integration problems, namely the *entity identification*[7] and *attribute value conflict*[8] problems. The first is the problem of identifying record instances from different export databases which correspond to the same real-world entity. The second arises when the attribute values in different databases, modeling the same property of a real-world entity, do not match. To resolve these two problems in the specification of global schema mapping, Myriad supports *outerjoin* and *generalized attribute derivation (GAD)*[9] as integration operations in addition to the usual set of relational operations.

Two-way outerjoin (denoted by $\overleftrightarrow{\bowtie}$) is particularly useful for merging a pair of relations such that tuples coming from different relations satisfying some predicate can be combined together. The result of outerjoin preserves all attributes of component local relations. Out-

---

* In case a local DBMS does not support the relational model, it is the gateway's responsibility to convert the non-relational schema to a relational schema.

erjoin integrates local relations in such a way that the tuples from different relations are combined when some predicate is satisfied. Those tuples that do not satisfy the predicate are still preserved by padding NULL values for attributes that do not exist in their relations.

While the integrated relation after outerjoin preserves all attributes from different relations, the desired global relation may only want to expose some attributes or show another set of attributes derived from local attributes. Myriad provides *global attribute derivation* functions (GAD) for users to specify the functions that resolve local attribute value conflicts and map them to global attributes.

GAD over a relation R is defined as:

$$
\begin{aligned}
GAD(R, \quad & (a_1 \quad F_1(X_1)) \\
& (a_2 \quad F_2(X_2)) \\
& \cdots \\
& (a_m \quad F_m(X_m)))
\end{aligned}
$$

where $F_1, \cdots, F_m$ are the attribute derivation functions for integrated attributes $a_1, \cdots, a_m$ respectively. For each i, X_i is the set of attributes from R that are used in F_i to derive a_i. GAD computes each tuple in the output relation from a tuple in R, and the integrated attributes of the output tuple by applying the attribute derivation functions on the R tuple.

Example

There are three local databases, namely DB_A, DB_B and DB_C, that maintain information about restaurants. A federated database FDB is defined over them, merging the common tuples between the local databases, and preserving tuples which are found only in one local database. The keys of the local relations have been underlined.

Local database DB_A:

RES_A	(<u>rname</u>,street,bldgno,phone,founder,rating,cost)
$MENU_A$	(<u>rname,foodname</u>,cuisine,chef,price, svctime,spiciness)

Local database DB_B:

RES_B	(<u>rname</u>,street,bldgno,phone,rating, cost,parking,delivery)
$MENU_B$	(<u>rname,foodname</u>,cuisine,country,price,svctime)

Local database DB_C:

RES_C	(<u>rname</u>,street,bldgno,phone,wdayhrs,wendhrs)
$BUFFET_C$	(<u>rname,mealtype</u>,costperhead,numofdishes)

Federated database FDB:

RES	(<u>rname</u>,street,bldgno,phone,founder, rating,cost,parking,delivery,wdayhrs,wendhrs)
$MENU$	(<u>rname,foodname</u>,cuisine,chef,country, price,svctime,spiciness)
$BUFFET$	(<u>rname,mealtype</u>,costperhead,numofdishes)

The derivations of RES, $MENU$ and $BUFFET$ in FDB are shown as algebraic expressions below*:

$RES \leftarrow GAD(\ \overleftrightarrow{\bowtie}(\{RES_A, RES_B, RES_C\},\quad (RES_A.rname = RES_B.rname)\ and$
$(RES_A.rname = RES_C.rname)\ and$
$(RES_B.rname = RES_C.rname))\ ,$

$\quad\quad\quad (rname \quad\quad\quad\quad F_{key}(RES_A.rname, RES_B.rname, RES_C.rname))$
$\quad\quad\quad (street \quad\quad\quad\quad F_{any}(RES_A.street, RES_B.street, RES_C.street))$
$\quad\quad\quad (bldgno \quad\quad\quad\quad F_{any}(RES_A.bldgno, RES_B.bldgno, RES_C.bldgno))$
$\quad\quad\quad (phone \quad\quad\quad\quad F_{any}(RES_A.phone, RES_B.phone, RES_C.phone))$
$\quad\quad\quad (founder \quad\quad\quad\quad F_{id}(RES_A.founder))$
$\quad\quad\quad (rating \quad\quad\quad\quad F_{avg}(RES_A.rating, RES_B.rating))$
$\quad\quad\quad (cost \quad\quad\quad\quad F_{max}(RES_A.cost, RES_B.cost))$
$\quad\quad\quad (parking \quad\quad\quad\quad F_{id}(RES_B.parking))$
$\quad\quad\quad (delivery \quad\quad\quad\quad F_{id}(RES_B.delivery))$
$\quad\quad\quad (wdayhrs \quad\quad\quad\quad F_{id}(RES_C.wdayhrs))$
$\quad\quad\quad (wendhrs \quad\quad\quad\quad F_{id}(RES_C.wendhrs)))$

$MENU \leftarrow GAD(\ \overleftrightarrow{\bowtie}(\{MENU_A, MENU_B\},$
$\quad\quad\quad (MENU_A.<rname, foodname> = MENU_B.<rname, foodname>)),$
$\quad\quad\quad (<rname, foodname> \quad F_{key}(MENU_A.<rname, foodname>,$
$\quad\quad\quad\quad\quad\quad MENU_B.<rname, foodname>))$
$\quad\quad\quad (cuisine \quad\quad\quad\quad F_{any}(MENU_A.cuisine, MENU_B.cuisine))$
$\quad\quad\quad (chef \quad\quad\quad\quad F_{id}(MENU_A.chef))$
$\quad\quad\quad (country \quad\quad\quad\quad F_{id}(MENU_B.country))$
$\quad\quad\quad (price \quad\quad\quad\quad F_{max}(MENU_A.price, MENU_B.price))$
$\quad\quad\quad (svctime \quad\quad\quad\quad F_{avg}(MENU_A.svctime, MENU_B.svctime))$
$\quad\quad\quad (spiciness \quad\quad\quad\quad F_{id}(MENU_A.spiciness)))$

$BUFFET \leftarrow GAD(\quad BUFFET_C,$
$\quad\quad\quad (<rname, mealtype> \quad F_{key}(BUFFET_C.<rname, mealtype>))$
$\quad\quad\quad (costperhead \quad\quad\quad F_{id}(BUFFET_C.costperhead))$
$\quad\quad\quad (numofdishes \quad\quad\quad F_{id}(BUFFET_C.numofdishes))$

Among the attribute derivation functions used in the GAD operations, F_{key} is defined as follows:

$$F_{key}(k1, k2) = \begin{cases} k1 & \text{if k1 is not NULL} \\ k2 & \text{if k1 is NULL but k2 is not} \end{cases}$$

F_{id} is the identity function. F_{any} selects any non-NULL value from its input arguments if there are any and outputs NULL value otherwise. F_{max} selects the largest value among its input arguments and outputs NULL value if all input arguments are NULL. F_{avg} performs average over its input arguments. Note that the above functions are chosen to be simple just to make the example easy to understand.

---

* For the equality (=) in outerjoin ($\overleftrightarrow{\bowtie}$) predicates, we assume NULL is equal to any non NULL value.
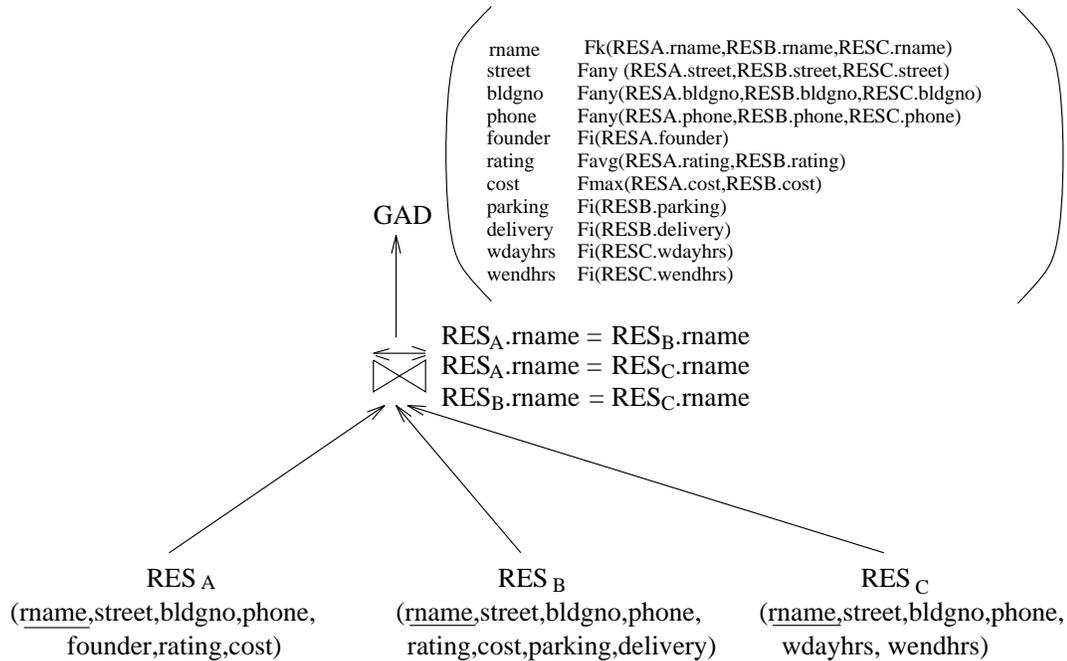
rname   Fk(RESA.rname,RESB.rname,RESC.rname)
street   Fany (RESA.street,RESB.street,RESC.street)
bldgno   Fany(RESA.bldgno,RESB.bldgno,RESC.bldgno)
phone   Fany(RESA.phone,RESB.phone,RESC.phone)
founder   Fi(RESA.founder)
rating   Favg(RESA.rating,RESB.rating)
cost   Fmax(RESA.cost,RESB.cost)
parking   Fi(RESB.parking)
delivery   Fi(RESB.delivery)
wdayhrs   Fi(RESC.wdayhrs)
wendhrs   Fi(RESC.wendhrs)

GAD

RES_A.rname = RES_B.rname
RES_A.rname = RES_C.rname
RES_B.rname = RES_C.rname

RES $_A$
(rname,street,bldgno,phone,
founder,rating,cost)

RES $_B$
(rname,street,bldgno,phone,
rating,cost,parking,delivery)

RES $_C$
(rname,street,bldgno,phone,
wdayhrs, wendhrs)

Figure 2. Derivation of integrated relation RES.

Currently, Myriad provides a set of system-defined attribute derivation functions which include some commonly used aggregate functions, e.g. $F_{min}, F_{max}, F_{avg}, F_{id}, F_{sum}, F_{key}$, etc. To reconcile more varied attribute value conflicts, Myriad allows federation DBAs to define attribute derivation functions using any programming language.

Global schema mapping

Global schema mapping defines the derivation of each integrated relation in a global schema from a set of related export relations. In Myriad, we represent the derivation of an integrated relation as an operator tree the leaf nodes of which denote the export relations involved, and internal nodes denote either relational or integration operations. The choice of a tree-structured representation greatly simplifies the subsequent step of augmenting global queries. In Figure 2, we show the operator tree of RES as defined in the example of the previous section.

Specifying the global schema mapping is an important but difficult task. The mapping process usually requires the local DBAs to co-operate among themselves as well as with the federation users. In Myriad the schema mapping information for various federated schemas is represented in files. To reduce the difficulty of generating these files we are currently developing a graphical input tool for the DBAs.

Loosely-coupled versus tightly-coupled FDBSs

The Myriad integration framework supports both loosely- and tightly-coupled FDBS.[10] In a *loosely-coupled* FDBS, the federation DBAs do not attempt to integrate export relations from different local databases. Federation users can freely use the data manipulation operations provided by the FDBS to integrate data from export relations in a manner that best suits their precise needs. An example of a loosely-coupled FDBS is the Multidatabase project.[4] Advantages of the loosely-coupled approach include: (a) no DBA effort is required to resolve semantic heterogeneity among local databases; and (b) this does not require the DBA to anticipate the needs of federation users. However, the loosely-coupled approach has the serious drawback of requiring federation users to be familiar with the locations and the contents of the heterogeneous export schemas in order to pose direct queries on them. In this approach, since there is no integrated schema, the amount of optimization that can be performed by the optimizer is also limited. In contrast, a *tightly-coupled* FDBS hides the location and semantic heterogeneity of local databases from the federation users by providing them with global schemas.

Myriad DBAs can define a global schema such that incompatibilities between export relations are resolved. Myriad users therefore see the global schema as an ordinary single database schema. Global queries on such a global schema can only involve relational operations. To achieve flexibility in integrating local databases, Myriad also allows a global schema to contain simply the export relations of the local databases without resolving their differences. While the location transparency of local databases is still supported, the Myriad users of this global schema define the exact way of merging these export relations in their global queries using the integration operators and relational operators provided by Myriad. Thus, the framework provided by Myriad supports both loose and tight integration in a uniform way.

SYSTEM ARCHITECTURE

This section documents the detailed design of Myriad's various architectural components. Figure 3 shows the different subsystems and components in Myriad. Each subsystem is responsible for a different functional layer of Myriad. Each component within a subsystem implements a different functionality of the subsystem. Figure 4 shows the Myriad process architecture.

A suite of *application tools* are used by Myriad users and DBAs to access the system. A user-friendly query formulator provides an interactive query interface to federated databases. Interactions among a collection of related global transactions, specified using synchronization constructs, are preprocessed into multiple global transactions and executed according to the specified inter-dependencies. DBAs use the schema integrator to merge the potentially conflicting local databases and to define the global schemas. Schema browsers allows both DBAs and Myriad users to comfortably familiarize themselves with the relations and attributes of the global schemas.

Components in the *query processing subsystem* support different aspects of query processing in Myriad. A *federated query manager (FQM)* manages global query processing for a Myriad user. There is one FQM process for each user active in the system. The FQM does the mapping from the user's view of the integrated database to the FDBS's view of the individual local databases. A *federated query agent (FQA)* is responsible for query processing at a local database that is participating in a global transaction. FQAs are told what
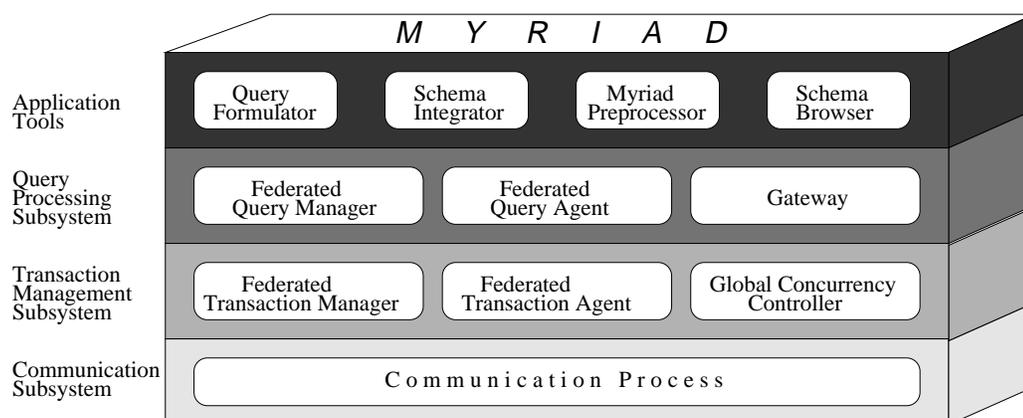
Figure 3. Myriad functional layers.

to do by site execution plans that are built and sent by the query's FQM process. Query processing in Myriad is done in a distributed fashion with FQA processes sending intermediate results to each other before one of them returns a single result to the FQM. *Gateway* processes are Myriad's interface to local databases. A gateway receives SQL requests from an FQA, translates it into the local DBMS's data manipulation language, and then submits the request to the local DBMS.

The *transaction management subsystem* components support transaction management in Myriad. The *federated transaction manager (FTM)* and *federated transaction agent (FTA)* are analogous to the FQM and FQA in the query processing subsystem. An FTM manages global transactions for a Myriad user. An FTA co-ordinates a global transaction's access to a particular local database. There is a different FTA for each local database participating in a global transaction. There is exactly one *global concurrency controller (GCC)* in the system. The GCC is responsible for ensuring the serializability of global transactions.

The *communication subsystem* consists of a *communications process (CP)* running at each physical site in the system. The CP routes messages and intermediate query results between logical and physical sites. It also serves as a site co-ordinator for each physical site. The CP does the mapping between logical and physical sites; it is the only process in Myriad that knows about physical sites. Each CP can communicate with all other CPs. Each CP also keeps track of which transactions are active in each of the logical sites at its physical site.

Design principles

We now describe the principles on which the design and implementation of the Myriad prototype has been based.

1. *Implement a clean design first, and then tune for performance.* Whenever we had the choice between a clean design and a more complicated (and may be more efficient) design, we chose the former. Should bottlenecks be found in our implementation, we felt it would be easier to improve performance after having done a clean initial design than it would be to implement a complicated design.
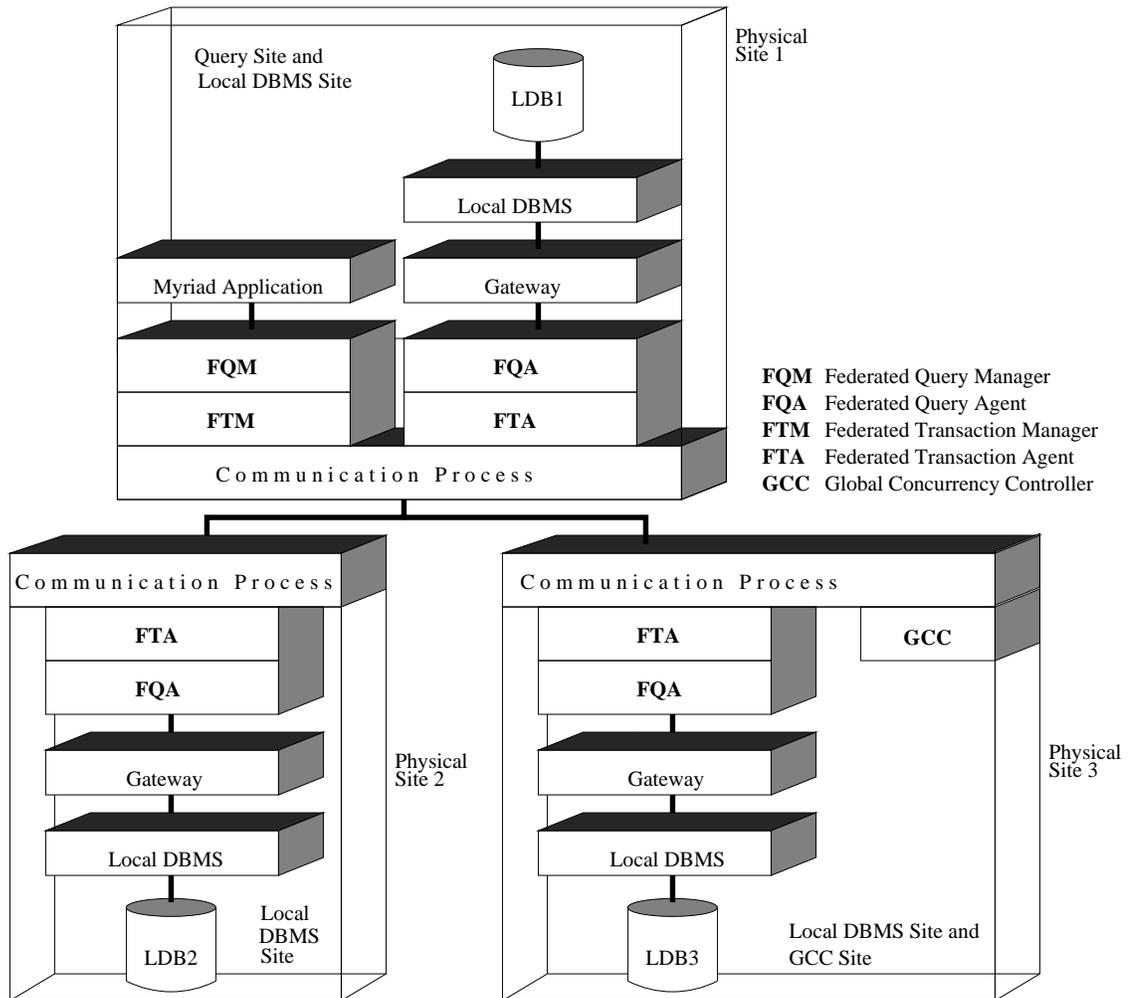
Figure 4. Myriad process architecture.

For example, the FQM and FTM have been implemented as separate processes be-
cause they perform separate functions: one handles global query processing and the
other global transaction management. However, they are both created and destroyed
at the same time, and they communicate frequently with each other. From a perfor-
mance perspective, it makes sense to implement them as one process even though
they perform two different functions. However, designing them as one large process
would have blurred that separation and most likely have resulted in a less flexible
implementation. We implemented them as separate processes and ended up with two
processes that could be merged into one without a significant loss of clarity. The same
argument applies to the FTA and FQA.

2. *Use an event-driven message-passing paradigm.* Myriad processes are state-based en-
tities which do some action and enter a new state when an event happens. The events
in Myriad are the reception of messages from some other process. Message passing is

an asynchronous activity and therefore allows significant flexibility in how different processes communicate with each other. Messages and their responses need not have one-to-one correspondence and therefore handling of asynchronous events such as local database aborts becomes easier.

3. *Restrict the number of processes that any one process can communicate directly with.* The different components in Myriad can be viewed as a logical pipeline, with the user at the top and the local database at the bottom. The FQM, FTM, CP(s), FTA, FQA, and gateway (in that order) make up the middle (see Figure 4). Any process along that pipeline can talk directly to its two immediate neighbors. If a process needs to send a message to another process that is more than one hop away in the pipeline then the message must be passed through each process between the two.

This restricted message passing model requires extra hops for some types of messages, but its simplicity actually reduces the overall message passing overhead in Myriad. It greatly simplifies the propagation of events and messages through the system and the handling of asynchronous events such as local database aborts and errors.

SYSTEM IMPLEMENTATION

In this section we describe the implementation of each Myriad subsystem and how the functionalities presented in our design is achieved. We have classified the description under the following headings: Query Processing and Optimization, Transaction Management, Communications and Process Management, and Application Level Tools.

Query processing and optimization

The Myriad query subsystem addresses the problem of processing and optimizing global queries in a *heterogeneous* and *locally autonomous* environment. Apart from having the data distributed, federated query processing is also affected by the heterogeneity of data among databases, and local autonomy among different database systems. These additional factors manifest themselves in the following ways:

1. *Processing of integration operations.* Due to data conflicts among databases, the federated query processor has to support new integration operations in addition to the standard set of relational operations. Currently, Myriad supports the outerjoin and GAD operations to resolve the entity-identification and attribute-value conflict problems. An important point worth noting is that regardless of whether the FDBS supports loosely- or tightly-coupled export databases, the federated query processor must be able to evaluate integration operations, as well as to optimize queries involving these operations.

2. *Local query execution autonomy.* In a FDBS, the local DBMSs involved are fully autonomous entities. Each local DBMS may adopt its own query processing strategy which in general can neither be revealed nor dictated by the federation software. Hence, query optimization is divided into global and local optimization phases. The former is performed by the FDBS while the latter by the local DBMSs.

3. *Availability of statistical information and local cost model.* The success of query optimization often depends on the accuracy of knowledge about the statistics on referenced relations and the execution cost model of the local DBMSs.[11] In a FDBS environment, some local DBMSs may not have, or may not supply (due to autonomy) sufficient

information about database statistics, DBMS workloads, and cost models. This may seriously restrict the opportunities for performing global query optimization. The design of Myriad query subsystem recognizes these limitations, and focuses on adopting useful heuristics to generate execution plans that reduce the amount of local data accesses. We also introduce an additional statistics collection step into query processing so that cost-based optimization can be performed.[12]

The following describes in detail the design and implementation of functional components in the Myriad query subsystem.

Gateway

Myriad gateways support a relational interface to local DBs using SQL. This design choice was made mainly because of the popularity of SQL gateways for non-relational DBMSs and relational but non-SQL DBMSs, e.g., IDMS/R,[13] Ingres,[14] etc. By adopting SQL as the uniform interface to all local DBMSs, we achieve portability and interoperability in the Myriad design.

Three kinds of database services are provided by Myriad gateways. They are *access control, transaction services*, and *query services*. *Access control* services include connecting to and disconnecting from export DBs with security features such as export DB names and passwords. Before any query can be performed, the appropriate export database name and its password must be provided. The gateway supports transaction services such as *begin-transaction*, *prepare*, *commit* and *abort*. These transaction services allow Myriad to manage global subtransactions as transactions in the local DBMS. To perform a retrieval with a Myriad gateway, both the query (in the form of an SQL SELECT statement) and the name of the file to contain the result relation, must be supplied. The gateway translates the SQL statement into local query statement(s), transforms the local query result into a relation, and populates the result file.

In cases where local DBMSs are relational and can participate in federated query processing, intermediate results may be created within the local databases during query processing. Hence, Myriad gateways support SQL DDL and DML statements such as table creation, deletion, and tuple insertion. In Myriad, we have designed the gateway such that it can create a temporary local relation when given an intermediate result file. Note that the 'bulk load' utilities available in the local DBMSs are normally not suitable for this purpose since the temporary relation must be associated with some global-subtransaction in order for them to be queried or removed within the subtransaction. Myriad gateways discard all temporary relations when the global-subtransaction is committed or aborted.

Federated query manager (FQM)

The FQM is responsible for generating efficient execution plans for queries submitted by the application layer. The execution plan determines the manner in which the federated query agents and gateways collaborate in order to produce the final result of the global query. Figure 5 depicts the functional components within a federated query manager. The meta-knowledge about the export databases and local DBMSs is also shown in the figure.

In Myriad the FQM process always exists together with a corresponding FTM process. These processes are alive throughout an interaction session with a user process. One user interaction session can last for any duration and may comprise many transactions. When
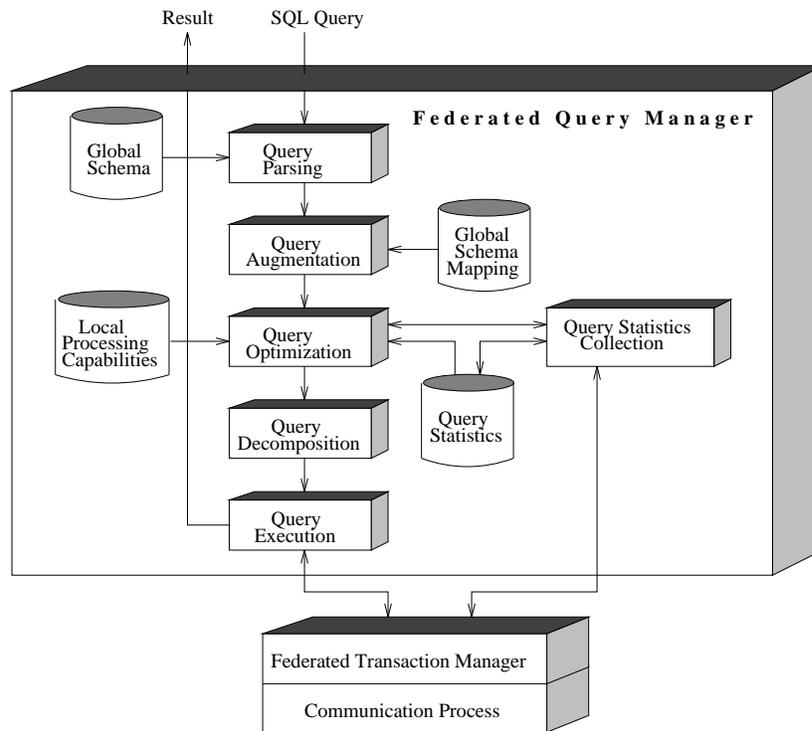
Figure 5. Federated query manager.

the FQM is created as a user session begins, it loads up the meta-data about the federated database from configuration files. These data include information such as the global schema and also processing capabilities of the component databases.

A query from the application layer is first parsed against the integrated relation definitions in the *global schema*. Any syntactic error or illegality in the query is detected and reported to the caller without further processing. The parser in Myriad has been generated using the Lex/Yacc utilities. A query that is parsed successfully is then represented as an operator tree with relational operators as internal nodes and integrated relations as leaf nodes. The query augmentation unit further replaces the leaf nodes by derivation expressions that correspond to the integrated relations. This step requires knowledge of the *global schema mapping* as discussed previously. The augmented operator tree contains both relational and integration operations.

The augmented operator tree is then passed to the Myriad query optimizer which generates an execution plan to efficiently produce an equivalent query result. The optimizer decides which operations in the query tree are performed at which FQAs and gateways. It also determines the order in which these operations are carried out. The optimization requires information about the local processing capabilities and query statistics such as export relation sizes, cost parameters, etc. In order to attain interoperability and preserve local autonomy, a federated query processor must be able to accommodate gateways and FQAs with different processing capabilities. In Myriad, we model the local processing capabilities of FQAs and gateways as sets of operations. Each FQA or gateway must make its supported set of

operations known to the federation so that it will not be asked to handle any unsupported operations during query execution. Since there is an extra overhead incurred for exchanging relations between gateways and FQAs, our optimizer gives preference to the FQA when an operation is supported by both the FQA and the gateway at the same site. Due to local autonomy, it is possible that export relation sizes may not be made known to the FQM. To perform a cost-based query optimization, the optimizer has to generate some query fragments for the query statistics collection unit. The statistics collection unit subsequently sends these query fragments to the appropriate sites to collect the required relation statistics. Note that the intermediate relations produced by the query fragments may be kept in the FQA while only the statistics are returned to the FQM.

Due to the use of outerjoin and GAD as integration operations, the existing algebraic transformation framework, for relational operations alone, is not sufficient for optimizing Myriad global queries. We have therefore developed an algebraic transformation framework involving outerjoin, GAD and other relational operations. The extended set of transformation rules explores new opportunities of query transformation by incorporating useful semantics about user-defined functions (of GAD operations) and query predicates (see Reference 9 for a detailed discussion).

Eventually, the FQM decomposes the optimized query execution plan into one or more *query fragments*. Each query fragment is assigned to either a FQA or gateway. All query fragments going to the same logical site, as well as the partial orderings of executing them, are compiled together as a *site execution plan*. The FQM sends all the site execution plans to the FTM which then distributes each site execution plan to the FQA at the appropriate logical site for execution.

Federated query agent (FQA)

The federated query agent co-ordinates the processing of global queries at a logical site. It also evaluates the integration operations and relational operations not supported by the gateway at the site. To execute the site execution plan assigned by the FQM, the FQA has to interact with both the local gateway and FQAs at other logical sites. Figure 6 shows the functional components within an FQA.

Similar to the FQM/FTM pair the FQA always exists at a local database (server) site in conjunction with an FTA. The FQA/FTA pairs are also associated with one user session, but they do not exist beyond the duration of a transaction. The FQA/FTA pair is created at all sites accessed by the transaction and when a transaction completes execution these processes are removed.

During query processing, each FQA receives a site execution plan containing query fragments and their execution order. A query fragment is ready for execution in either the gateway or the FQA if all the input relations it requires are available at the site. An FQA employs a site execution monitoring unit which keeps track of the execution readiness of every local query fragment. Whenever an intermediate result is produced locally or is shipped from a remote site, the readiness status of the local query fragments are updated. A query fragment is evaluated as soon as it is ready for execution. In this manner, parallelism in query execution is achieved among all FQAs and gateways participating in a global query.

If an FQA query fragment is ready for execution, it is given to the FQA query fragment evaluation unit. This unit can evaluate user-defined attribute derivation functions by forking processes to execute the appropriate user programs. If a gateway query fragment is ready for
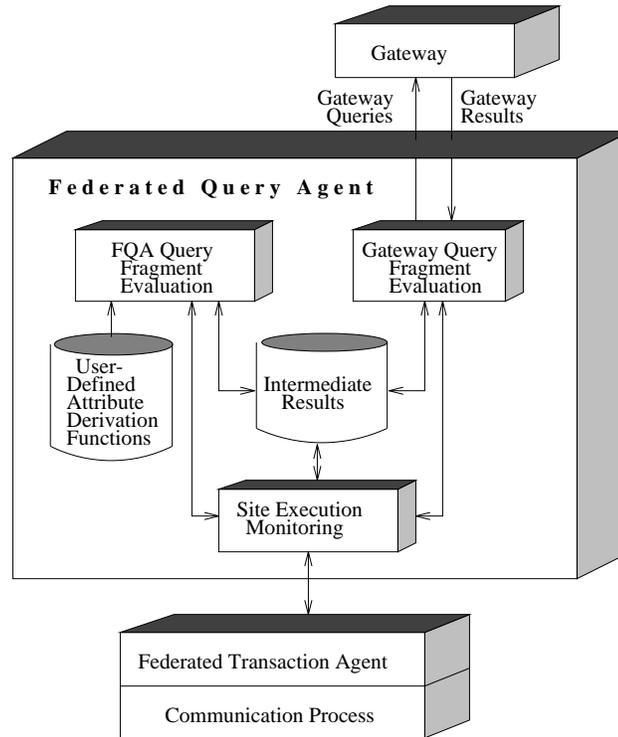
Figure 6. Federated query agent.

execution, the gateway query fragment evaluation unit (within the FQA) translates it into an SQL SELECT statement before sending the query to the gateway. Upon the completion of a query fragment evaluation, the site execution monitoring unit determines whether: (i) any other local query fragment is ready for execution; (ii) the result has to be shipped to another FQA for further execution; or (iii) shipped to FQM as the final result.

In the current Myriad design, it is assumed that information required for integration, e.g., attribute derivation functions in GAD operations, can be provided to all the FQAs. This assumption will be relaxed in the future to allow more flexibility in the FQA execution.

Transaction management

One of the most important features supported by database management systems is *transaction management*. Most of the existing DBMSs provide some transaction facility. However, even though each component DBMS is able to guarantee consistent local execution, without further control undesired results may occur with respect to global transactions that access data across sites through the FDBS. We describe such a scenario in the following example.

Example. The Smiths have one account at bank A and another at bank B. An FDBS has been built on top of the databases of banks A and B that allows users to access data in databases of both banks. Mr Smith issues a global transaction G_1 that transfers \$10,000 from account A to account B. Meanwhile Mrs. Smith issues another global transaction G_2

to examine the balances of both accounts. G_1 and G_2 are as follows:

G_1: G_2:
$account_A \leftarrow account_A - 10,000$ print($account_A$)
$account_B \leftarrow account_B + 10,000$ print($account_B$)

Without any control, the following execution may happen:

bank A: bank B:
$account_A \leftarrow account_A - 10,000$ print($account_B$)
print($account_A$) $account_B \leftarrow account_B + 10,000$

Obviously, the above execution is undesirable. Myriad controls the execution of global transactions in such a way as to prevent anomalous results. The correctness criterion we adopt for the execution of global and local transactions is *serializability*. Traditional approaches used in distributed database systems for concurrency control are not applicable in an FDBS, due to the unwillingness (in general) of local DBMSs to provide internal control information (due to autonomy) and the differences in transaction management mechanisms adopted by local DBMSs (due to heterogeneity). Over the past five years, transaction management has become one of the most active research areas in FDBSs. Many algorithms have been proposed.[15,16,17,18,19,20,21,22,23,24] Only recently have researchers started to investigate the relative performance of the various proposed algorithms.[25] However, such investigation is still in its early stage, and it is not clear how different algorithms behave in a real system. Therefore, one of our design choices in transaction management was to make our architecture flexible to allow various algorithms to be implemented and tested.

Myriad contains three types of processes for handling transactions: federated transaction manager (FTM), federated transaction agent (FTA), and global concurrency controller (GCC). Each FTM exclusively controls a global transaction. A global transaction issues queries that access data controlled by multiple DBMSs. The queries of a global transaction that access the same site are grouped together as a subtransaction which is treated as a single transaction by the local DBMS. A FTA monitors the execution of a subtransaction. The global concurrency controller (GCC) verifies whether the current execution is consistent. The FTMs, the FTAs and the GCC work co-operatively to achieve serializable execution. The detailed operations of each type of components depend on the chosen concurrency control protocol.

Concurrency control

The proposed concurrency control algorithms for FDBS can be classified on the following two dimensions:

1. Centralized versus distributed control
2. Restricted versus general transaction model

Most of the proposed algorithms use centralized control, see, e.g., References 15–22 and 24. GCC process is responsible for monitoring the execution of global transactions. A centralized concurrency control algorithm can be realized by the GCC. Through the co-operation between GCC, FTM and FTA, consistent execution is guaranteed. For those algorithms that employ distributed concurrency control (see, e.g., References 23 and 26),

each of them can be realized by the co-operation between FTM and FTAs. GCC, in this case, is not used.

Many of the proposed algorithms put restrictions on the global transaction model so as to achieve consistent execution or prevent global deadlocks. For example, some algorithms require a global transaction to declare the sites it will access at the *begin-transaction* statement (see, e.g., References 16, 19, and 22). Others require no value dependency between subtransactions (see, e.g., References 23, 24, and 27). While the applicability of restricted transaction models depends on the kinds of federated DB applications, our transaction management components can realize any of the restricted transaction models as well as the standard transaction model, in which no such restrictions are imposed.

In the current Myriad implementation, we adopt the standard transaction model (as currently used by many transaction processing systems). This assumption will be relaxed in the future. By laying out the processes as we have, the transaction management components – FTM, FTA, and GCC – can directly communicate with each other without involving the query processing mechanism. Realization of different transaction management algorithms, in Myriad, therefore involves only modifications to the TM modules. Since the services offered to the query processing components are the same, the changes in the TM modules do not affect the former. This approach of associating functionality with a module also illustrates the flexibility of Myriad as a testbed.

Recovery

Myriad assumes that each local DBMS has the ability to recover from failure. However, even though each local DBMS can guarantee consistent local executions in the presence of failure, the execution of global transactions, which access data across several local DBMSs, may reach an inconsistent state when failure occurs. To prevent the anomalous execution results caused by failure, some researchers argue that all local DBMSs should provide *prepare to commit* state to support the two-phase commit (2PC) protocol. In the real world, however, the local DBMS may or may not support such visible *prepare to commit* state for global subtransactions. Myriad uses a 2PC protocol over all local DBMSs. In case a local DBMS does not support *prepare to commit* state, the associated gateway simulates it by logging the write operations after a subtransaction enters its *prepared* state and re-submitting these operations if failure occurs. Several algorithms proposed in the literature (see, e.g., Reference 28) can be used to control the resubmission.

However, the simulated 2PC mechanisms do not come for free. Several restrictions have to be imposed, including local concurrency control mechanisms (e.g., only strict 2PL is allowed), local transaction data access (e.g., the data set is partitioned into two parts that are updatable by local and global transactions, respectively) or local recovery procedure (e.g., global subtransactions have exclusive use of local DBMSs after recovery). While these restrictions may compromise some autonomy, we see them as a price to pay in order to achieve consistent execution in the presence of failure.

Deadlock handling

Many of the proposed concurrency control algorithms cannot prevent global deadlocks. Therefore, some mechanisms to detect and resolve global deadlocks are required. Unlike traditional distributed database systems, FDBSs cannot obtain the exact data conflict relationship among global transactions due to autonomy of participating databases. Thus, time-

out is needed to establish the *potential* conflict relationship among global transactions.[29,30] A timeout period is associated with each query submitted to the local DBMS. If the result of a query does not return within the timeout period, the global transaction to which the query belongs is assumed to conflict with some other global transactions executing at the same local DBMS.

An optimal timeout period is determined by several factors, such as query size, degree of data contention, degree of resource contention, resource availability, etc. Myriad uses a mechanism that dynamically adjusts the timeout period based on the current system performance. Specifically, we record the system performance and measure the average response times for transactions during different intervals of time. The timeout period is then selected to allow most of the transactions to complete. If the system performance increases as a result of this adjustment, the timeout period is further tuned in the same direction. However, if the performance decreases then a correction is done in the opposite direction.[31]

Communication subsystem and process management

The communications process (CP) in Myriad serves several purposes. It acts as a co-ordinator for distributed processing and manages the inter- process communication at different physical sites. Myriad can be started by a DBA from any of the participating sites in the system. This site provides the startup files containing the system configuration, as well as the global schema information to Myriad.

The startup site then invokes the communication process at each participating sites and passes on the configuration information as well as the meta-data. All process interaction in Myriad is done using logical site IDs, transaction IDs and query IDs. CPs at each site are the only processes aware of the physical site to logical site mapping. Hence all communication between processes which are likely to be at different physical sites is done through the CP.

We have used the Unix message queues[32] for communication between processes at the same physical site and stream sockets for inter-site communication in Myriad. Each process type (e.g., the FTM or FQA) has one designated incoming mailbox for messages. All processes at invocation will attach to the message queue for its corresponding process type. Message destination in a queue is identified by using the process-id of the receiver as the message type.

Process management is another function carried out by the CP. Every process in Myriad registers itself with the CP using its transaction-id, logical site-id and process-id. Messages send across physical sites specify their destination as a combination of the logical site-id, transaction-id, and process type. Since there is a unique correspondence between this address and a process-id in the CP process tables, the CP can route the messages to their correct destination.

System shutdown is also co-ordinated by the communication process. The shutdown process can be initiated from any of the participating sites. Shutdown command is propagated to the CPs at all the sites and the local CPs remove the message queues and terminate themselves. Any other running processes also will terminate themselves when they determine that the message queues have been removed. Since Myriad has been implemented based on a event-driven paradigm where message receptions are the events, we could terminate the processes in a very efficient way as described above.

Application tools

To simplify user and DBA interaction with Myriad we have developed a set of application tools. These include a X/Motif based user interface (UI), a schema browser, and a schema integrator tool.

The user interface provides a windows-based environment for the user to conduct the session with Myriad. The facilities include specification of transactions and queries, and options to view or save the results files. A user session is started up when the interface is invoked. This allows the UI to attach to the message queues and start communicating with its corresponding FQM process. During a session, the user can start transactions which will bring up the query window. Queries may be posed interactively or read in from files. Query results are displayed and the user is given options for saving the result files. Transactions may be aborted or commited.

Schema browser is a very useful tool with which the user or DBA can look up the global schema mapping of the federation. The browser has options to display the derivation trees for each of the global relations as well as the operations being done at each node in the tree. Figure 7 shows the use of schema browser to view the global relation *RES* and the attribute derivation function for the field *RNAME*.

A schema intergrator tool which is currently being developed will allow the DBAs to graphically construct the global schema information without having to get buried in the intricacies of the global schema representation. This tool will also aid in making changes to the global schema.

IMPLEMENTATION AND TESTING EXPERIENCE

This section describes those experiences in implementing and testing our software which we feel are unique to the Myriad project. During the design and implementation phases, we made some important decisions that had an impact on the architecture as well as construction of the system. In the following, we briefly describe the rationale for each decision and its impact.

1. *Use peer-to-peer model rather than client–server model for the communication between Myriad processes.* The client–server model is currently widely used by as an interface for communicating with other systems. However, client–server model incurs extra messages and requires synchronization between client and server processes. We believe that the peer-to-peer model is the right choice for intra-system communication. The peer-to-peer model requires neither acknowledgements nor waiting for each request transmitted, which is flexible and efficient.
2. *Use message queues for intra-site process communication.* To realize the peer-to-peer model, we choose *message queues*, supported by UNIX System V, for process communication within a site. A message queue is associated with each type of processes. A process retrieves a message from its associated message queue and handles this message.
3. *Use event-driven paradigm to describe the behavior of each process.* As described before, each process retrieves a message from a specific message queue, takes some action according to the requirements of the message, and then gets the next message for processing. This entire procedure is best realized by a *state diagram*. A state diagram consists of a set of states and transitions between states that are caused by events. In
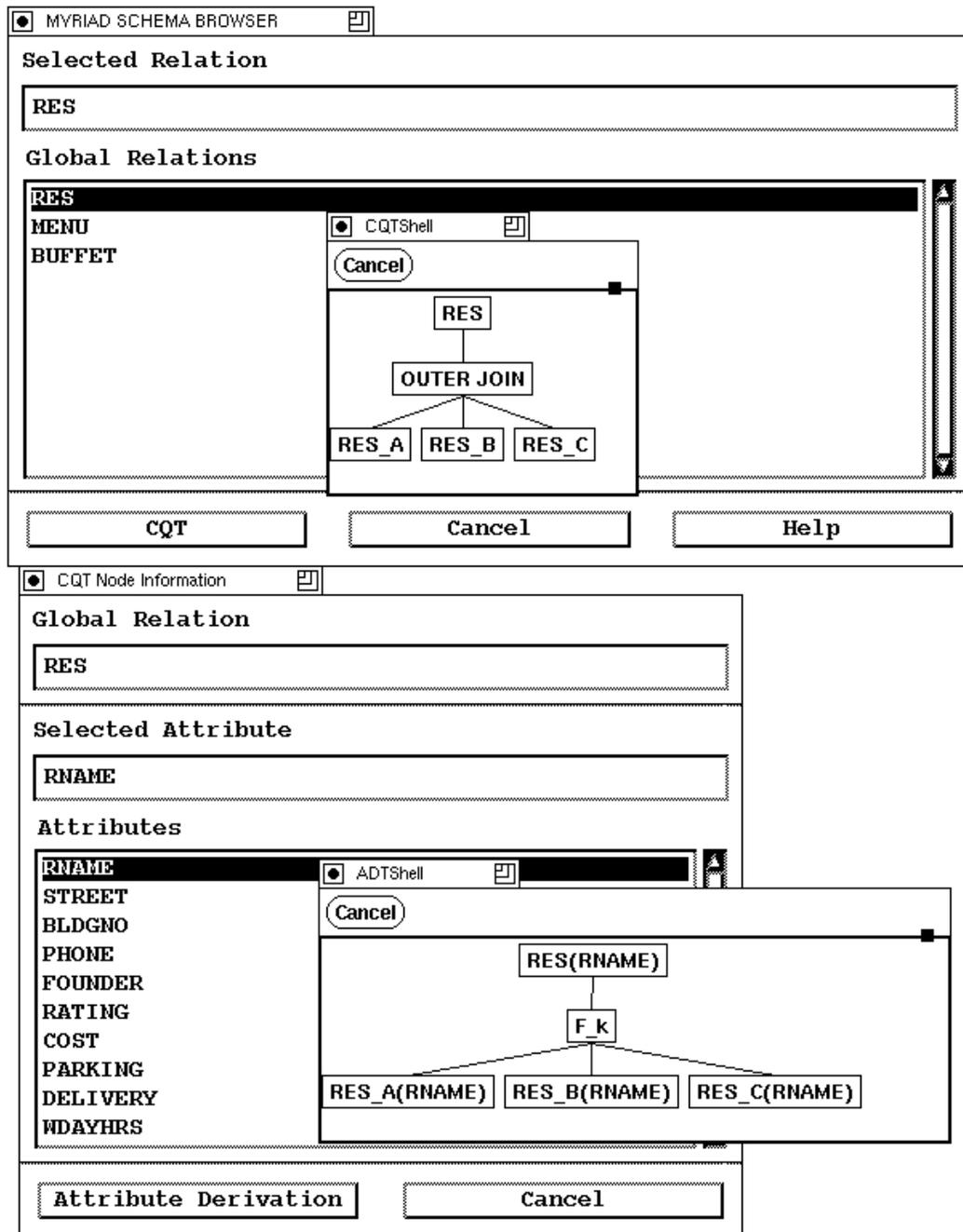
Figure 7. Viewing a global schema using the browser.

Myriad implementation, events are characterized by the commands specified on the messages in message queues. Figure 8 shows the state diagram of FTM.

State diagrams allow us to examine the message flow at the design level. This is especially useful when several processes are involved, as in the Myriad system. Sometimes the designers of different processes may have inconsistent assumptions concerning message exchange. We found several such inconsistencies when walking through the state diagrams of different processes, especially since a team of six people was working on the project.
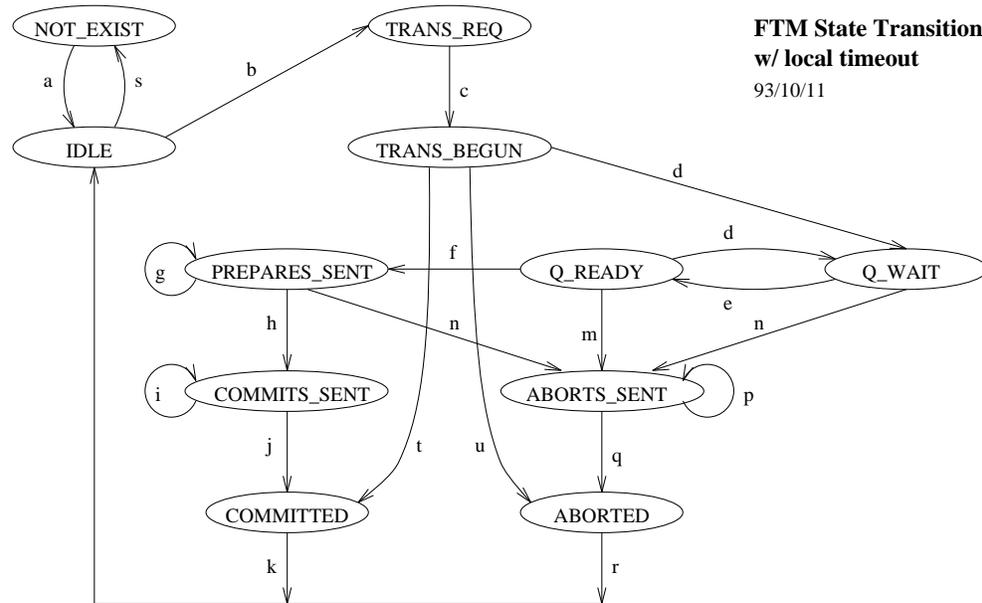
Software engineering principles

In addition to using standard coding conventions and mechanisms for debugging, we have developed a mechanism for directly translating from state diagrams to an implementation. *Standard translation of a state diagram to code*: As mentioned before, we use a state diagram at the design level to describe a process. While state diagrams provide a convenient means by which the designers can examine consistent message exchange, we need a straightforward mapping between the state diagram on the paper and the corresponding code in the system.

We need a data structure to record the information of a state diagram and a function associated with the data structure that, given an incoming message and the current state, invokes an appropriate procedure (to take the appropriate action). That procedure may bring the process to another state according to the state diagram. The following shows the data structure used to store the state diagram of 'ftm', described in the previous section.

```
std_Private _sma_Table ftmSmaTableGto = {
    /*--                                  S T A T E            */
    /*--                    */ { _ABORTED,
    /*--                      */ _ABORTS_SENT,
    /*--                    .   */ _COMMITS_SENT,
    /*--                    .   .   */ _COMMITTED,
    /*--                    .   .   .   */ _IDLE,
    /*--                    .   .   .   .   */ _NOT_EXIST,
    /*--                    .   .   .   .   .   */ _PREPARES_SENT,
    /*--                    .   .   .   .   .   .   */ _Q_READY,
    /*--                    .   .   .   .   .   .   .   */ _Q_WAIT,
    /*--                    .   .   .   .   .   .   .   .   */ _TRANS_BEGUN,
  /*-- M E S S A G E        .   .   .   .   .   .   .   .   .   */ _TRANS_REQ},
 {{MSG_FTM_ABORT_TRANS,     {dis,dis,inv,inv,dis,inv,inv,a05,inv,a07,inv}},
  {MSG_FTM_ABORT_TRANS_ACK, {inv,a10,inv,inv,inv,inv,inv,inv,inv,inv,inv}},
  {MSG_FTM_BEGIN_TRANS,     {inv,inv,inv,inv,a15,inv,inv,inv,inv,inv,inv}},
  {MSG_FTM_BEGIN_TRANS_ACK, {inv,inv,inv,inv,inv,inv,inv,inv,inv,inv,a20}},
  {MSG_FTM_COMMIT,          {inv,inv,inv,inv,dis,inv,inv,a25,inv,a27,inv}},
  {MSG_FTM_COMMIT_ACK,      {inv,inv,a30,inv,inv,inv,inv,inv,inv,inv,inv}},
  {MSG_FTM_DEREGISTER_ACK,  {a35,inv,inv,a40,inv,inv,inv,inv,inv,inv,inv}},
  {MSG_FTM_EXEC_QUERY,      {dis,dis,inv,inv,dis,inv,inv,a45,inv,a45,inv}},
  {MSG_FTM_EXEC_QUERY_ACK,  {dis,dis,inv,inv,inv,inv,inv,inv,a50,inv,inv}},
  {MSG_FTM_EXIT,            {inv,inv,inv,inv,a60,inv,inv,inv,inv,inv,inv}},
  {MSG_FTM_LOCAL_ABORT,     {dis,dis,inv,inv,inv,inv,a70,a05,a80,inv,inv}},
  {MSG_FTM_PREPARE_ACK,     {inv,inv,inv,inv,inv,inv,a75,inv,inv,inv,inv}}
  }
};
```

The associated look up function will take this data structure, the current state, and the type of incoming message, and invoke a corresponding procedure. For example, if the current state is _Q_READY and the incoming message is MSG_FTM_EXEC_QUERY, procedure 'a45()' will be invoked.

**FTM State Transition
w/ local timeout**
93/10/11

NOT_EXIST   TRANS_REQ

a   s   b   c

IDLE   TRANS_BEGUN   d

d

g   PREPARES_SENT   f   Q_READY   Q_WAIT

e

n

h   n   m   n

i   COMMITS_SENT   ABORTS_SENT   p

j   t   u   q

COMMITTED   ABORTED

k   r

Label	Event (sending process in parenthesis)	Action
a	FQM forks FTM	get FQM pid, FTM pid, attach message queues
b	get MSG_FTM_BEGIN_TRANS (FQM)	register with CP, send MSG_GCC_BEGIN_TRANS
c	get MSG_FTM_BEGIN_TRANS_ACK (GCC)	save transaction ID, send MSG_FQM_BEGIN_TRANS_ACK
d	get MSG_FTM_EXEC_QUERY (FQM)	send MSG_FTA_EXEC_QUERYs to each logical site in the site list for the query. add each logical site to list of sites this transaction is at.
e	get MSG_FTM_EXEC_QUERY_ACK (FTA)	send MSG_FQM_EXEC_QUERY_ACK
f	get MSG_FTM_COMMIT (FQM)	send MSG_FTA_PREPAREs to each logical site this transaction is active at.
g	get MSG_FTM_PREPARE_ACK (FTA) or MSG_FTM_LOCAL_ABORT (FTA)	record that the sending site has responded to the prepare and how it has responded.
h	get last MSG_FTM_PREPARE_ACK (FTA) and no sites aborted	send MSG_FTA_COMMITs to each logical site in the transaction.
i	get MSG_FTM_COMMIT_ACK (FTA)	record that the sending site has committed its subtransaction
j	get last MSG_FTM_COMMIT_ACK (FTA)	tell CP to deregister FTM & transaction (send MSG_CP_DEREGISTER)
k	get MSG_FTM_DEREGISTER_ACK (CP)	send MSG_FQM_COMMIT_ACK, clear site list and transaction ID
m	get MSG_FTM_ABORT_TRANS (FQM) or MSG_FTM_LOCAL_ABORT (FTA)	send MSG_FTA_ABORT_TRANSs to each logical site in the transaction
n	get last MSG_FTM_PREPARE_ACK (FTA) or MSG_FTM_LOCAL_ABORT (FTA) and at least 1 site aborted	send MSG_FTA_ABORT_TRANSs to each logical site in the transaction
p	get MSG_FTM_ABORT_TRANS_ACK (FTA)	record that the sending site has aborted its subtransaction.
q	get last MSG_FTM_ABORT_TRANS_ACK (FTA)	tell CP to deregister FTM & transaction (send MSG_CP_DEREGISTER)
r	get MSG_FTM_DEREGISTER_ACK (CP)	send MSG_FQM_ABORT_TRANS_ACK, clear site list and trans ID
s	get MSG_FTM_EXIT (FQM)	send MSG_FQM_EXIT_ACK and then kill itself
t	get MSG_FTM_COMMIT (FQM)	tell CP to deregister FTM & transaction (send MSG_CP_DEREGISTER)
u	get MSG_FTM_ABORT_TRANS (FQM)	tell CP to deregister FTM & transaction (send MSG_CP_DEREGISTER)

Figure 8. State diagram of FTM.

This mechanism provides a straightforward mapping between a state diagram and its implementation. It dramatically reduces the implementation and debugging effort.

We also used other tools for system implementation and management, such as RCS for code and document version control and xdbx for module testing. As these tools are quite straightforward and widely used in software development, we will not repeat our experience of them here.

Lessons learned

The implementation decisions and software engineering principles we have described gave us some advantages. However, we also found some disadvantages in applying these decisions and principles. The following lists the lessons we have learned.

1. State diagrams and their mapping to code greatly reduced our effort in debugging the message exchange among processes. By comparing the state diagrams of several processes, we discovered many problems in the initial design of process message exchange. This enabled us to correct the design before coming to the code level. The mapping of state diagrams to code is very straightforward and we seldom found any problems in code if the state diagrams were correct.
2. While a standard coding convention kept our code clean, uniform and easy to maintain, it also placed some burden on our programmers. In fact, some conventions may not be that useful. For example, our naming convention carries a lot of information, some of which, e.g., including token type information in the name, is seldom used and can be eliminated. Besides, because of the elaborateness of the coding standard, some programmers did not follow it completely. Thus we feel that there is a need to simplify the coding convention and to get consensus among programmers before adopting it.
3. In contrast to the coding standard, we feel that we need more information about the debugging message. The current debugging routines only show the process type a message is from. Because Myriad may involve multiple sites, we often need to know which site this message is from. Besides, the timing of a message can be important. Sometimes we may want to know the order of occurrence of messages from different sites. More information will help the programmers locate problems with ease.

Implementation status

An implementation of Myriad has been realized in the UNIX environment involving a network of Sun SPARCstations. Currently, we have built gateways on two local DBMSs, namely Oracle and Postgres. To demonstrate the essential features of database integration in Myriad, we have constructed several example databases on both Oracle and Postgres such that relations from these databases are merged into integrated relations using outerjoins, GADs, and other relational operations. The prototype allows user-defined functions to be used in GAD operations. All programs are written in C and embedded query languages. The Myriad communication process is implemented using the BSD socket libraries.

In addition to local DBMS dependent gateways, the FQM and FQA components in Myriad query processing subsystem have already been implemented. The present FQM can parse any SELECT-PROJECT-JOIN statement in SQL and generates a set of site execution plans for all local DBMS sites involved. A fully-fledged query optimization module has not yet been implemented.
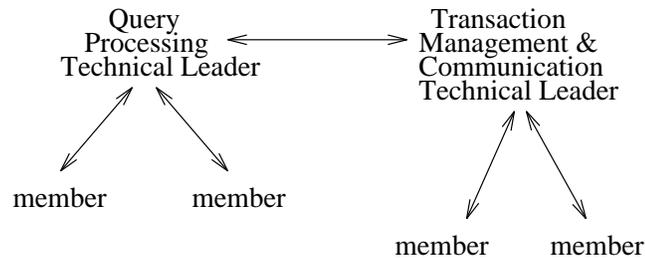
Figure 9. Myriad development team organization.

We have implemented a Myriad transaction management subsystem that supports the standard transaction model. Presently, the transaction management subsystem supports two-phase commit over local DBMSs in order to guarantee global serializability. To integrate those DBMSs that do not provide a visible *prepare-to-commit* state, Myriad currently simulates it without imposing any restriction on the autonomy of the DBMSs. That is, the gateway will simply return 'Yes' when it receives a *PREPARE-TO-COMMIT* request. In this case, global serializability is not guaranteed if failure of a prepared transaction occurs.

At the application tool level, an easy-to-use user interface has been implemented. This allows federation users to pose transaction as well as query requests to the Myriad system. We have also built a schema browser which can be used to view the global schema and the integration operations from a set of local export schemas.

PROJECT MANAGEMENT AND SCHEDULE

Project management has played a major role in the planning, organizing and managing of the Myriad prototyping activities. The project schedule is presented showing the ordering of various prototyping activities together with their deadlines. We also describe our experience as the result of adopting some of the project management principles and problems faced when adhering to the project schedule.

Team organization

At the time of initiating the Myriad project, the development team consisted of three Ph.D. students and three master students. As shown in Figure 9, the project team was organized in a hierarchical structure. The role of the faculty member has been to provide the overall research guidance and oversee the administrative matters. The prototype development team was divided into two technical sub-teams, each managed by a technical leader. Each technical leader was responsible for assigning tasks, conducting reviews and walkthroughs, detecting problem areas, and balancing the workload. The leaders also interacted frequently with each other in order to keep themselves informed about the progress, and to exchange technical information.

Code walkthrough

As part of an effort to verify and validate the quality of the Myriad prototype, we have adopted code walkthrough throughout the prototype life cycle, with emphasis given to the Myriad component design and program codes. Since the size of the Myriad system is moderate, our walkthroughs usually involved all the team members. The objectives were to allow team members to review the assumptions, decisions, and techniques adopted in the design or software development. The walkthroughs also ensured that the coding standards were observed.

Group communication

In the Myriad project, we conducted meetings at least on a weekly basis in order to update the team on the progress, to exchange technical problems and solutions, and to discuss some administrative matters. In some meetings, walkthroughs and software demonstrations were included. The frequency of meetings increased during the coding phase when many engineering problems required immediate attention and solutions. In addition to regular meetings for the entire team, small meetings between a technical leader and team members were also arranged on a one-to-one basis to discuss specific technical issues before presenting them to the rest of the team.

Documentation process

Documentation is viewed as an important activity in the Myriad prototyping effort. We share a common belief that without a proper documentation of our design and coding, it will be difficult for us or future team members to understand, maintain and extend the system. A set of design documents for query processing, transaction management and communication have been written and maintained throughout the project. We began with design documents with little details. They were constantly updated by the team members whenever the design became became more detailed or underwent revision. In order to facilitate the documentation process, all design documents are kept in a common directory and are managed by the *RCS* version control utility.

Project schedule

Figure 10 depicts the planned project schedule when the Myriad project was initiated. According to this schedule we had expected to design the Myriad architecture first, then design, implement and test of each functional subsystems, and finally integrate the subsystems and test the entire system. We were able to keep up with the planned schedule in the development of the subsystems. Testing of the subsystems, however, took longer than expected since we had to develop some tools which helped simulate communication with other subsystems. As the result of this delay the system testing and integration also ran one month behind the planned schedule and we were able to finish the project by March 1994.
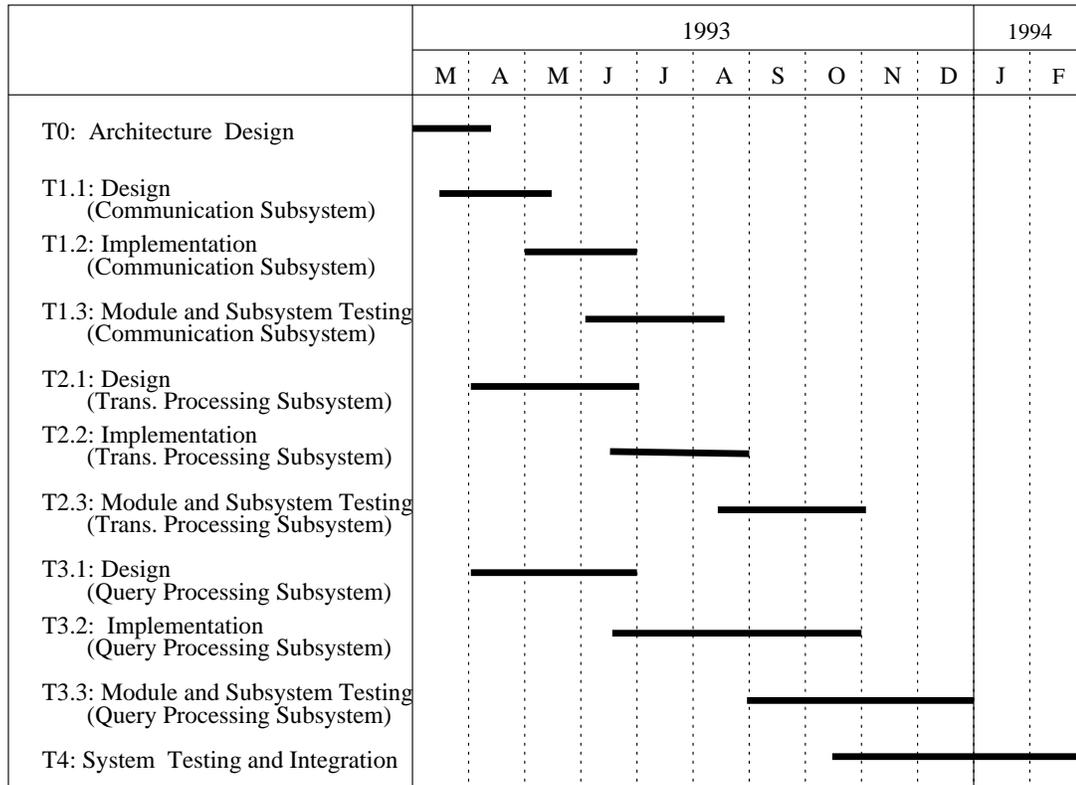
	1993									1994	
	M	A	M	J	J	A	S	O	N	D	J	F
T0: Architecture Design												
T1.1: Design (Communication Subsystem)												
T1.2: Implementation (Communication Subsystem)												
T1.3: Module and Subsystem Testing (Communication Subsystem)												
T2.1: Design (Trans. Processing Subsystem)												
T2.2: Implementation (Trans. Processing Subsystem)												
T2.3: Module and Subsystem Testing (Trans. Processing Subsystem)												
T3.1: Design (Query Processing Subsystem)												
T3.2: Implementation (Query Processing Subsystem)												
T3.3: Module and Subsystem Testing (Query Processing Subsystem)												
T4: System Testing and Integration												

Figure 10. Planned project schedule.

Code distribution

The Myriad project has so far produced about 35,000 lines of C code. As we develop more applications and implement various algorithms for performance comparison, the size of the system is expected to grow. Table 1 shows the current code-size distribution for different components of the Myriad system. It is clear that most of the code written was for the query processing subsystem. This is due to the large number of activities to be performed in query processing, e.g., query parsing, query augmentation, optimization, execution monitoring, etc. Furthermore, each different local DBMS requires a different gateway to be implemented, thus making the size of query processing subsystem grow with the number of local DBMSs. The SQL parser was generated using Lex and YACC systems. The transaction management subsystem does not require a lot of code since transaction processing activities are currently relatively simple. Upon adding recovery features, this component is expected to grow much more. The user interface, schema browser, and schema integrator tools have all been implemented in X/Motif. We had developed some tools, during the implementation, to perform module testing. These tools, as well as the standard functions used for providing debugging information are grouped into the miscellaneous category.

Experience

During the entire Myriad prototyping effort, the practice of project management principles has played an important role in structuring the development activities of our project members. We initially expected that our research knowledge in the federated database area would enable us to complete Myriad with little difficulty. Nevertheless, during the numerous design walkthrough sessions on both the design and coding, we discovered many technical problems which were not identified in our initial study. For example, during the implementation of the Postgres gateway, we realized that certain data types supported by the SQL standard do not exist in the Postgres DBMS. In particular, the fixed length character string and decimal data types are not supported by Postgres. This problem was later resolved by augmenting Postgres with user-defined data types as well as their operations. As part of its query processing steps, Myriad requires temporary relations to be created within the local DBMSs. To do so, the query processing team originally intended to use the relation import facility provided by local DBMS. It was during a walkthrough session that our transaction management members pointed out that such a design would violate the global transaction model because importing a relation would be treated as a separate transaction by the local DBMS. Consequently, we have modified the FQA and gateway designs so that they could allow temporary relations to be created and discarded within a single global transaction. Several other problems were discovered during the walkthroughs. In retrospect, we realize that walkthroughs have helped us tremendously in revealing unexpected design problems and enforcing coding standards. In building an advanced prototype liked Myriad, walkthroughs have helped implementation faults to be avoided at an early stage by incorporating feedback from all the other members. By reviewing others' design and codes, we have also gained a better understanding of the entire system.

Design documentation has been a time consuming but important activity in the Myriad project. We realized that the design has been constantly changing throughout the development process as we gained more experience and knowledge. In addition, many design and implementation decisions have been made during the meetings. Without detailed documentation, it would be impossible to recall the reason for making these decisions. Furthermore, as the membership of the Myriad development will change in the future (given the nature of a University environment) the documentation provides the continuity of knowledge and is an essential reading material for new members.

As already mentioned, we have built some tools for performing module testing. These tools helped us simulate the missing functional components and allowed us to perform white box testing.[33]

Lastly, we note that the project schedule has helped us greatly in monitoring the progress

Table I. Code distribution.

Myriad subsystem	Lines of C code
Query processing	11,700
SQL parsing	6000
Transaction management	2800
Communications	3000
User interface	2000
Schema browser	2000
Schema integrator	2000
Miscellaneous	5000

of our development efforts. However, we also experienced some difficulties in keeping our actual development exactly on schedule. For example, instead of completing the architecture design by mid-April, we made a major revision sometime in June as the reult of the need to simplify the interactions between FTM and FQM. We believe that the exploratory nature of the Myriad system made it necessary for us to revisit and modify previous design decisions. Therefore, we conclude that the planned project schedule is more of a tool to gauge the status of the project. It also kept us aware of the activities that are yet to be performed.

CONCLUSIONS AND FUTURE DIRECTIONS

Local autonomy and heterogeneity, at *both* system and database levels, characterize the nature of FDBSs. In this paper, we have presented the design and implementation experiences with Myriad, an FDBS prototype, which provides an environment with which to meet the increasingly important demand for the integration of existing autonomous database systems. The Myriad system architecture is flexible enough to allow the implementation of various transaction management and query processing strategies. Various research groups have been, in recent times, looking into the design of different query processing strategies and transaction management methods for the FDBS environment. We believe that a testbed such as Myriad can play an important role in validating and comparing these research results in a realistic setting. In the next phase of Myriad research, we will implement and evaluate a number of query processing strategies and transaction management algorithms.

We are currently integrating an application called *data flow query language (DFQL)*[34] with the Myriad framework. This tool has been designed to work as a visual data flow query interface to a database system, especially suited to science and engineering users. Integration of DFQL with Myriad will allow the DFQL users to issue queries on multiple databases simultaneously through a federated schema. Initial experience with this project show that Myriad design lends itself very well to the integration of new applications. A sucessful integration of this tool with Myriad will validate the flexibility of its architecture.

In query processing, we will pursue the development of a new query optimization strategy that handles the integration operations supported by Myriad. We will also study the problems of integrating both legacy applications and database systems in Myriad. We are currently implementing a temporary table creation and update facility in Myriad for applications in order to better utilize the capabilities of Myriad. This will later be expanded to allow updates to both the global and local tables.

In transaction management, Myriad research will address the issue of supporting restricted transaction models for different kinds of specialized global applications. A restricted transaction model may allow global serializability to be supported without sacrificing the autonomy of local systems. In situations where global serializability is not strictly required, we will examine the possibilities of using weaker correctness criteria for global transactions. Future enhancements of the Myriad prototype will also include a dynamically adjustable timeout mechanism for resolving global deadlock problems.

We have presented our experience in applying some software engineering techniques and project management principles to the implementation of the Myriad prototype. We believe our experiences and lessons learned will prove helpful to future system designers embarking on similar ventures.

REFERENCES

1. G. Thomas, G. R. Thompson, C.-W. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox and B. Hartman, 'Heterogeneous distributed database systems for production use', *ACM Computing Surveys*, **22**(3), 237–266 (1990).

2. C. Batini, M. Lenzerini and S. B. Navathe 'A comparative analysis of methodologies for database schema integration', *ACM Computing Surveys*, **18**(4), 323–364 (1986).

3. U. Dayal, *Query Processing in Multidatabase Systems*, Springer-Verlag, New York, 1985, pp. 81–108.

4. W. Litwin and A. Abdellatif, 'Multidatabase interoperability', *IEEE Computer*, **19**(12), 10–18 (1986).

5. R. Ahmed, P. D. Smedt, W. Du, W. Kent, M. Ketabchi, W. A. Litwin, A. Rafii and M-C. Shan, 'The Pegasus heterogeneous multidatabase system', *IEEE Computer*, **24**(12), 19–27 (1991).

6. O. A. Bukhres, J. Chen, W. Du and A. K. Elmagarmid, 'Interbase: An execution environment for heterogeneous software systems', *IEEE Computer*, **26**(8), 57–69 (1993).

7. E.-P. Lim, J. Srivastava, S. Prabhakar and J. Richardson, 'Entity identification problem in database integration', *Proc. 9th IEEE Data Eng. Conf.*, 1993.

8. E.-P. Lim, J. Srivastava and S. Shekhar, 'Resolving attribute incompatibility in database integration: An evidential reasoning approach', *Proc. 10th IEEE Data Eng. Conf.*, 1994.

9. E.-P. Lim, J. Srivastava and S.-Y. Hwang, 'An algebraic transformation framework for multidatabase queries', *Distributed and Parallel Databases, An International Journal*, to appear (1995).

10. A. P. Sheth and J. A. Larson, 'Federated database systems for managing distributed heterogeneous, and autonomous databases', *ACM Computing Surveys*, **22**(3), 183–236 (1990).

11. P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie and T. G. Price, 'Access path selection in a relational database management system', In *Proc. ACM SIGMOD Conference*, 1979, pp. 23–24.

12. E.-P. Lim and J. Srivastava, 'Query optimization/processing in federated database systems', in *Conference of Information and Knowledge Management*, 1993.

13. J. Martin, *IDMS/R: Concepts, Design, and Programming*, Prentice Hall, Englewood Cliffs, NJ, 1990.

14. D. Simonson and D. Benningfield, 'Ingres gateways: Transparent heterogeneous sql access', *Data Engineering Bulletin*, **13**(2) (June 1990).

15. Y. Breitbart, A. Silberschatz and G. Thompson, 'Update mechanism for multidatabase systems', *IEEE Data Engineering*, **10**(3), (1987).

16. Y. Breitbart and A. Silberschatz, 'Multidatabase update issues', in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1988.

17. C. Pu, 'Superdatabases: Transactions across database boundaries', *IEEE Data Engineering*, **10**(3), (1987).

18. C. Pu, 'Superdatabases for composition of heterogeneous databases', in *Proc. 4th Int. Conf. on Data Engineering*, 1988.

19. A. K. Elmagarmid and W. Du, 'A paradigm for concurrency control in heterogeneous distributed database system', In *Proc. 6th Int. Conf. on Data Engineering*, 1990.

20. Y. Leu and A. K. Elmagarmid, 'A hierarchical approach to concurrency control for multidatabases', in *Proc. 2nd Int. Symposium on Databases in Parallel and Distributed Systems*, 1990.

21. D. Georgakopoulos, M. Rusinkiewicz and A. Sheth, 'Serializability of multidatabase transactions through forced local conflicts', in *Proc. 7th Int. Conf. on Data Engineering*, 1991.

22. S. Mehrotra, R. Rastogi, H. F. Korth and A. Silberschatz, 'The concurrency control problem in multidatabases: Characteristics and solutions', in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1992.

23. R. K. Batra, M. Rusinkiewicz and D. Georgakopoulos, 'Decentralized deadlock-free concurrency control method for multidatabase transactions', in *Proc. 12th Int. Conf. on Distributed Computing Systems*, 1992.
24. S.-Y. Hwang, J. Huang and J. Srivastava, 'Concurrency control in federated databases: A dynamic approach', in *Proc. 2nd Int. Conf. on Information and Knowledge Management*, 1993.
25. J. Huang, S.-Y. Hwang and J. Srivastava, 'Concurrency control in federated database systems: A performance study', in *Proc. 7th Int. Conf. on Parallel and Distributed Computing Systems*, Las Vegas, Nevada, 1994.
26. J. Huang, S.-Y. Hwang and J. Srivastava, 'Distributed forward optimistic concurrency control for federated database systems', *Technical Report*, Honeywell Technology Center, 3660 Technology Drive, Minneapolis, Minnesota, 1992.
27. W. Du and A. Elmagarmid, 'Quasi serializability: A correctness criterion for global concurrency control in interbase', in *Proc. 15th Int. Conf. on Very Large Data Bases*, 1989.
28. S.-Y. Hwang, J. Srivastava and J. Li, 'Transaction recovery in federated autonomous databases', *Distributed and Parallel Databases, An International Journal*, **2**(2), 151–182 (1994).
29. Y. Breibart, W. Litwin and A. Silberschatz, 'Deadlock problems in a multidatabase environment', in *Proc. COMPCON*, 1991.
30. P. Scheuermann and H. Tung, 'A deadlock checkpointing scheme for multidatabase systems', in *Proc. 2nd Int. Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, 1992.
31. S.-Y. Hwang, J. Srivastava and J. Huang, 'Incorporating admission control into concurrency control in federated databases', *Technical Report* 93-56, Dept. Computer Sci., Uuniversity of Minnesota, MN, 1993.
32. W. Richard Stevens, *Unix Network Programming*, Prentice Hall, Englewood Cliffs, NJ, 1990.
33. B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1983.
34. B. S. Tjan, L. Breslow, S. Dogru, V. Rajan, K. Rieck, J. R. Slagle and M. O. Poliac, 'A data-flow graphical user interface for querying a scientific database. in *1993 IEEE Symposium on Visual Languages*, Norway, August 1993, IEEE Computer Society, pp. 49–54.