FINAL REPORT

to

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

on

RESIDENT DATABASE INTERFACES TO THE DAVID SYSTEM, A HETEROGENEOUS DISTRIBUTED DATABASE MANAGEMENT SYSTEM

Marsha Moroh
Department of Computer Science
The City University of New York
130 Stuyvesant Place
Staten Island, New York 10301
The College of Staten Island

NASA Grant NAG 5-763

December 15, 1988

Table of Contents

1 INTRODUCTION 1

2 Resident Database Interfaces What is an Interface? 2

3 Work Performed 7
3.1 Interface Specifications 7
3.2 General Interface Software 7
3.3 Interfaces 8
3.3.1 ORACLE. 8
3.3.2 INGRES. 8
3.3.3 FITS and General Files. 8
3.3.4 Other Relational DBMSs. 8
3.3.5 Other Interfaces. 9

4 Summary and Suggestions for Further Research 9
4.1 Build more interfaces 9
4.2 Generalize the interface-building process 9
4.3 Generalize the interface-building process 10

1 INTRODUCTION

The focus of this research project has been the development of a methodology for building interfaces of resident database management systems to a heterogeneous distributed database management system under development at NASA, the DAVID system. A secondary goal of the research was a demonstration of the feasibility of that methodology by construction of the software necessary to perform the interface task.

The first section of this document contains the interface "terminology" developed in the course of this research. The second section describes the work performed; the third section summarizes the results, puts the work into the context of the DAVID system as a whole, and contains suggestions for further research.

We feel that this research project has been very successful in meeting its goals in two ways. First of all, we now know HOW to build interfaces to heterogeneous distributed database management systems: we have isolated the component parts that make up an interface, and we know how those parts fit together, and how they interact with the host system. Secondly, the immediate practical application of our research was the construction of several of the interfaces (some by us, some by others according to the guidelines developed in this research), and the construction of the software which "hooks" those interfaces into the DAVID system. The DAVID system, with its associated interfaces, has been demonstrated on several occasions at the Goddard Space Flight Center. These demos have generated considerable enthusiasm on the part of the scientists for whom communication is facilitated by the enhanced database capabilities these interfaces can provide.

2 Resident Database Interfaces

What is an Interface?

An interface between database management systems is a layer of software which allows queries and transactions made through one DBMS to be processed on another, without replication of data and without conversion of either database. Without such an interface, if scientists who have data on 2 different DBMSs want to share information, one of them would have to convert his/her data to conform to the standards of the other's database management system.

The database management systems that are to be interfaced will be referred to as resident DBMSs. By resident DBMS, we mean any database management system currently in use as a commercial product or a research tool. When interfaced, these resident DBMS's are connected via a single DBMS which is a heterogeneous distributed system. All queries and transaction requests pass through this central system, and so by installing an interface between any resident DBMS and the heterogeneous distributed DBMS, we provide that resident with the capability of communicating with any other resident so interfaced. We will call the heterogeneous distributed database management system for which all these interfaces will be built, the host DBMS. It is not necessary for the host and the resident to be on the same physical machine.

What are the components of an Interface?

For every DBMS to be interfaced, a package must be installed on the host DBMS. The package contains the following components:

Generation of a Database Definition

There are two routines for generating database definitions (sometimes called schemas) in an interface package. The first takes a resident database definition and produces the corresponding host database definition. The second produces the resident database definition, given the host definition. The former is used to install an existing resident database onto the host system. (See Installing a Resident Database, below.) The latter routine enables a user to create a definition for a new resident database through the host DBMS without knowing the syntax of the resident. (See Defining a Resident Database, below.)

Defining a Resident Database through the Host DBMS

A user can DEFINE a new resident database through the host DBMS. This is accomplished in two operations. The first, creating the actual database in the resident DBMS, requires the generation of a set of commands which log on to the resident DBMS, and present it with a definition (or schema) for the new database to be

created. The creation of that schema, in turn, requires the host-to-resident definition generator described above. The second operation in the DEFINE process is the installation of the database definition of the resident in the database directory of the host.

Installing a Resident Database

Before host commands can be issued to process a resident database, the resident must be INSTALLed on the host system. The INSTALL operation is used to connect existing resident databases to the host, thereby establishing interfacing capability. In the Installation routine, there is no reading of data; the data remains in the resident DBMS and only the database definition is put in the directory of the host DBMS. To extract the definition from the resident, commands must be issued to enter the DBMS and generate a listing of the schema of that resident. Then, using the resident-to-host definition generator, (described above), a host version of the resident definition is derived. This form of the database definition can then be put into the directory.

Deleting a Resident Database through the Host DBMS

A resident database may be DELETEd by a command through the host DBMS. Deletion is a two-step process. First, the actual database is deleted from the resident DBMS. Then, the corresponding directory entry for that resident database is deleted from the host DBMS. To delete the resident database, commands must be generated which log on to the resident DBMS, provide the authorization to delete, and perform the deletion.

"Deinstalling" a Resident Database from the host DBMS

When a DEINSTALL command for a particular resident database is issued, all record of that database is dropped from the host's directory; the resident database itself is left intact. The purpose of the DEINSTALL is to break the connection between the resident and the host. Hence, it is the opposite of the INSTALL operation. The DEINSTALL operation does not appear in succeeding chapters, because the entire operation is performed by the host; no resident participation is necessary.

Once the necessary information about the resident has been installed in the directory of the host DBMS, queries and transactions meant for the resident can be submitted through the host. This is done in the next group of routines. The interaction between the resident and the host can take place on 3 possible levels: the query language level, the table level, and the path level.

The Query Language Level

Some resident DBMSs support their own query languages. For those DBMSs, a query or transaction involving a resident submitted through the host DBMS can be translated by the resident interface into a query or transaction in the language of the resident, and then executed by the resident in its own environment. The following primitive queries have been isolated as the components of any complex query between a resident database and a host database:

Generalized Selection-Projection. A selection-projection or selection-multiprojection is performed on a resident database. The results of the query are stored in a new database on the host.

Semijoin. A join is performed between a resident database and a host database. The result is a new database on the host, and a table of pointers to rows of data items in both the source host database and the new result database.

Store-to-Database. A selection-projection is performed on a database on the host DBMS; the result is stored in a new database in the resident DBMS.

Insert, Delete, Update. Transactions submitted through the host are performed on a resident database. Insert adds row(s) of data items, delete removes a row or more, update modifies a row or rows.

The Path Level

If the resident DBMS has a high-level language interface (such as C, PASCAL or FORTRAN), and supports a command to retrieve information from several tables of the database as a single access (as is often the case in a hierarchical database), then requests for information and transactions submitted via the host can be handled by the host at the Path Level. The host DBMS determines the proper access path through the resident database; then calls on the path access routines of the resident DBMS to navigate through the resident database. These routines are:

Path First Row. The first path "row" of the resident database (i.e., the first row of every table that makes up the specified path through the database) is read by the resident DBMS, and the data inserted into the host DBMS data buffers. There is one host DBMS data buffer for each corresponding table row of the resident. Any boolean evaluation is performed by the host DBMS on the data in the buffers.

Path Next Row, Path Previous Row, Path Last Row. The required path "row" of the database is read into the corresponding host DBMS buffers, where any boolean evaluation is done.

Path Insert, Update, Delete. A path "row" of the database, including all of the tables specified in the path, is inserted, updated or deleted.

Path Assign. This routine performs "housekeeping" functions necessary to allow the resident DBMS to communicate with the host at the path level: it determines which tables of the database must be used to make up the path, and allocates necessary data buffers required by the resident to contain path information.

Path Deassign. This routine performs any "housekeeping" functions necessary to terminate a resident interface path access by disassociating the resident with that path if necessary, and deallocating any special data areas set aside by Path Assign.

The Table-Row Level

In some cases, it is necessary to access the resident DBMS at the lowest level, i.e., the Table-Row Level. This table-at-a-time access capability is used for those resident DBMSs which support a high-level language interface, such as FORTRAN or C, but provide neither a query language nor access at the path level. This is common in network DBMSs. In these cases, the only way for the host DBMS to process the resident data is at the single table level. When resident DBMSs support user access at the table-row level, requests for information and transactions submitted via the host are processed by the host until the request is reduced to a call for a single row of a single table. At that level, the request can be handled by the resident DBMS. If the table-row request includes a boolean, the boolean evaluation is done by the host on the data after it has reached the host buffer. The table-row routines are:

Table-Row First, Table-Row Last. The first (last) row of a table of the resident database is read into a buffer of the host DBMS. Any boolean evaluation is done in the host buffer.

Table-Row Next, Table-Row Previous. The next (previous) row of a table of the resident database is read into a buffer of the host. Any boolean evaluation is done in the host buffer.

Table-Row Insert, Table-Row Delete, Table-Row Update. A row of data of a table in the resident database is inserted, deleted or updated.

Table-Row Connect. A row of data which has just been inserted into a table in the resident database, is connected to the proper parent. This routine only has applicability in a database of network structure.

Table-Row Disconnect. A row of data is disconnected from the designated parent, and if that is the last parent, the row is deleted from the database. This routine only has applicability in a database of network structure.

Table-Row Parent. Given a row of a table, and the name of a parent table of that table, this routine returns a row of data from the parent table. This routine only has applicability in a database of network structure.

Assigning and Deassigning Resident Databases. Before either table-row routines or path routines can access a resident database, certain initialization functions must be performed. Similarly, after path or table-row access routines to a resident have been performed, some termination operations functions must take place before the resident DBMS is exited. These routines are as follows:

Assign Database. This routine performs "housekeeping" functions necessary to allow the resident to communicate with the host: it establishes the necessary data buffers and variables needed by the resident, logs on to the resident, provides the necessary security, performs any necessary language translation or execution of query language statements.

Deassign Database. This routine performs "housekeeping" functions necessary to terminate host interaction with a resident: closes the resident database, logs off of the resident DBMS, and deallocates any special data areas set aside by Assign Database.

3 Work Performed

The following is an overview of the work performed under this grant, and the progress made towards solving the interface problems.

3.1 Interface Specifications

A set of detailed specifications for 11 database management systems commercially available, and for general file systems (see 3.3.3, below) were developed, according to the design outlined in section 2, above. There were two levels to these specifications: for each major module, the high-level spec explains what it does and how it connects to the rest of the system, while the low-level spec provides details at the algorithm level. Each low-level spec is accompanied by an example. The complete set of specifications is available at NASA's Goddard Space Flight Center (contact Dr. Barry Jacobs, code 634), or from the author.

These specifications were intended for use by people building interfaces to the DAVID system: in particular, the author and a set of university "interface-building" teams funded by the DAVID project. The author supplied each team with a set of specifications for the particular interface the team was working on, and provided technical support for the teams while the interface building was going on. This was particularly important in the initial stages of the construction of the DAVID system, when the system design was changing almost daily. An important part of the author's task was to insulate the team from all of the changes by providing them with a constant environment, and by building connecting software which handled the changes in DAVID's design.

3.2 General Interface Software

One of the modules of the DAVID system is the software which enables queries and transactions submitted to DAVID to be channeled to the proper resident DBMS interface, and the results transferred back. This "interface to the interfaces" was developed by the author in the course of the research. It consists of a set of routines which accept the query or transaction, parse it, put it into a data structure designed by the author, and, after determining which resident interface should be the recipient, sends it to the proper place. When the transaction is complete, the software transmits the results back to the DAVID system.

The data structure used in the development of the interfaces was created by the author for the university inter-face-building teams to insulate them from the changes constantly occurring in the software design of DAVID at the time, and to spare them from the agony of having to deal with

a parser and parse tree which were extremely general, obscure, and changing daily. This data structure and the routines which filled it in were intended to be a temporary measure to be used only until DAVID stabilized; however, it proved to be so useful that it is still in place today.

3.3 Interfaces

Once the complete specifications were in place (see 3.1, above) and the general interface software (see 3.2) was written, several interfaces to commercial DBMSs currently being used at NASA, were written and installed into the DAVID system. These interfaces are described briefly below:

3.3.1 ORACLE.

The ORACLE interface was designed and developed at University of Houston according to our specifications, and installed at NASA on a VAX machine.

3.3.2 INGRES.

The INGRES interface was developed at Louisiana State University according to our specifications, and installed at NASA on the AT&T 3B2. A subsequent version for the VAX was developed at NASA.

3.3.3 FITS and General Files.

Interfaces to both the FITS (Flexible Image Transfer System) file system and to general files with no headers were designed, developed and installed by the author. These are not, strictly speaking, database management systems, but instead are file systems whose data are read via user-constructed programs. Nevertheless, it was felt by the author that such interfaces would prove extremely useful to the scientific community, since they would provide these files with database management system facilities. These facilities include query processing capability and, via the DAVID "reading room" facility, the ability to perform random accesses and transactions on the data.

3.3.4 Other Relational DBMSs.

Interfaces to several other relational DBMSs were built at NASA recently, according to our guidelines. It is a telling observation that, although the first (ORACLE) interface took almost two years to develop (beginning in 1985) while our guidelines were being developed, the IMDM interface (one of the relational DBMSs used at NASA) took just a few weeks to implement, in the fall of 1988.

3.3.5 Other Interfaces.

A few other interfaces were begun at the early stages of this research. Two of them, RBASE and KNOWLEDGEMAN, microcomputer relational DBMSs, were developed by the author as the guidelines were being developed. These two were never interfaced to DAVID because of the fact that the DAVID system does not yet run on a DOS machine (and maybe never will!). Two more, IDMS (under development at Brooklyn College) and RAMIS (being developed at University of Toledo) were never finished. The author is continuing to work with a student from Brooklyn College on the IDMS interface. Although this interface can not be completed at this time (since the intersection of the hardware supporting DAVID and the hardware supporting IDMS is null), it is particularly interesting to us because it is a network-structured DBMS, which allows us to test theories about interfacing these kinds of DBMSs to a host. Similarly, the RAMIS interface is a good example of a hierarchical DBMS, and hence there is a lot to be learned about interface techniques from its implementation. The author has applied for a local (CUNY) university grant to continue the hierarchical investigations on FOCUS, a DBMS available on a variety of computers.

4 Summary and Suggestions for Further Research

Now that the properties of an interface have been isolated, detailed specifications for interfaces spelled out, and the software to connect the interfaces to DAVID installed, we feel that a respectable start has been made on the task of interfacing different resident databases to a heterogeneous distributed host. But much remains to be done. Some of these activities are summarized below:

4.1 Build more interfaces

Now that the interface-building process is clearly understood, the task of building an interface is not so formidable as it was in the past. There are a number of DBMS systems currently being used by NASA scientists which could now be interfaced to the DAVID system. This will enable those users to be able to exchange information without having to do data conversion (something which they are not now able to do).

4.2 Generalize the interface-building process

We have been investigating the possibility of reducing the task of building interfaces to that of building a set of static, data-independent templates which get filled in at query- (or transaction)- processing time by a set of software routines common to all interfaces. If this approach works,

then the construction of an interface will require no programming at all -- just the creation of a set of templates (one for each interface component).

4.3 Generalize the interface-building process

The task of constructing the templates described above, may be able to be performed by an expert system, which would interactively solicit requirements and characteristics of the candidate DBMS; then construct the set of templates by altering a "master template" to make it DBMS-specific. In that case, the interface builder doesn't need to know anything about the DAVID system at all, and needs to know only about his/her own resident DBMS being interfaced.