MULTI SYSTEM DATABASES

By Rolf Frydenberg
Hewlett-Packard Norge A/S
Oslo, Norway

SUMMARY

The term Multi System Databases (or MSDB, for short), is used to refer to applications that utilize data from more than one database, located on geographically separated computers.

The paper presents and discusses MSDB, or distributed data sharing systems as they are sometimes referred to. This area is still largely experimental, but a major R & D effort is going on. MSDB will probably become of major importance within the next decade, and at least by the turn of the century.

The first part of this presentation focuses on the end-user needs that make MSDB a future necessity. Distributed data sharing systems based on homogeneous computers and database management systems are discussed, as well as those based on heterogeneous systems.

The second major part of this paper presents two existing (though mostly experimental) MSDB's. We have also included a brief prensentation of HP3000 capabilities in this area.

Finally, we indicate the direction we think this area is moving in, and how we - as users - can prepare for that development.

I. WHAT IS A MULTISYSTEM DATABASE?

Before proceeding to the details of this presentation, I would like to give you a thumbnail sketch of what multisystem databases are. This should bring the area into closer perspective, so that we all know what we are looking at.

A multisystem database is a database that consists of information stored on more than one computer system. The data stored may be of different formats, or the same. And we will also look at any file - flat file, KSAM file, or "true" database - as being a database or part of one; so long as a description of its format is available to us.

The computer systems that store the multisystem database may be identical - e.g. a set of HP3000's - or they may be different - e.g. some PC's, some HP3000's and a non-HP mainframe.

The data communications link between the computers is assumed to work, but we will not specify it in any level of detail: It might be a dial-up asynchronous link, a leased 9600 bps line, an X.25 connection, or an SNA-link. For the sake of this presentation, we will ignore those differences.

Let me also mention that there are two other expressions often applied to multisystem databases: Distributed databases, and distributed data sharing systems. We will treat these expressions as referring to the same genereal concept, though some people do consider at least the term "distributed databases" as being more restrictive that the other two terms.

There are many possible approcaches to distributing a database over a network of computer systems. Four of the main distribution strategies are:

1. **Independent databases;** this means that each computer system has its "own" database(s), controlled fully by this computer system. Each independent database "contributes" some of its data to the overall Multisystem database.
2. **Centralized database with replicated subsets;** this means that there exists one, centrally located, database, which contains all the data. Subsets of

this database are copied to other locations, mainly for higher speed of access to frequently used data. Typically, updates are only accepted at the central site, and transmitted to remote locations in batches.

3. **Horizontal partitioning;** this means that the same record types may exist at many locations, but a specific record is at just one location. A file of orders, for example, could be distributed so that all orders are located at the warehouse that will process them.
4. **Vertical partitioning;** this means that different components of a specific record may be located at different sites. In a vertically partitioned database, some kind of replication may be used, so that all locations may know all customer-numbers and customer-names, but more detailed information is only stored at the site that actually handles this customer.

In many real-life situations the best solution is a combination of strategies. Strategy number 1 - independent databases - is the most common method of handling the integration of previously independent databases. That makes this strategy important now, though not necessarily as important in the future.

Another aspect of multisystem databases is whether they allow purely local users to exist. Local users are those who access data on one specific computer in the network, without knowledge of, or access to, the "complete" mulitsystem database. Some MSDB systems allow this - particularly those systems designed to handle existing databases - whereas others treat all users as global users, and all transactions as global transactions.

Processing global queries (i.e. data retrieval from the multisystem database) is a reasonably straight forward operation. Updates, though are potentially much more complex. There are two reasons for this complexity: If data is replicated, then all copies of the record must be updated; if the record is split between multiple sites, then all sites must update their part of the record. And, of course, the MSDB's system must keep track of whether the individual updates succeeded, and if they did not, then it must perform whatever backtracking is necessary.

If there is no replication within the multisystem database, then it is possible to restrict all updates to being purely local transactions, i.e. transactions that may be processed completely on a single computer system. This significantly simplifies the manner in which updates are processed. Centralized systems also simplify updating the database, since only the central copy of the data must be updated.

II. THE NEED FOR MULTISYSTEM DATABASES

A large amount of research into the area of multi system databases is currently being carried out all over the world. Is this research just being carried out "for the fun of it", as the business world often accuses the academic community of, or is it something we will really need in the not too distant future?

It is my contention, that multi system databases are something we need, and consequently that it is an important topic for research. And though it may take a number of years before complete, high-performance, easy-to-use systems are available, we need to forge ahead in this research. There may be many intermediate solutions that represent small steps in the right direction.

Since this is not intended as a theoretical paper - I am not a theoretical computer scientist, but rather a practical engineer - let me present the kind of situations where I think multi system databases can represent a solution for users.

Many data processing users currently have more than one computer system. This is true not only of large corporations with central mainframes, but also for medium-sized companies that may have two or more minicomputers; or smaller

companies where the computer mix is made up of microcomputers only, or micros combined with minis. In all these cases, corporate data is stored on more than one computer.

These, often diverse, computer systems communicate more and more closely. This need for communication has grown out of the need for access to data through terminal-emulators, but that is only a brief stage in the total development of corporate data communications: As the storage capacity of "non-central" computers (micros and minis) continues to grow rapidly, so the need for accessing this data directly, instead of transferring it to a central site and accessing it there, grows proportionately.

For many users this is already in the process of becoming a problem. The amount of data stored at the decentralized sites is growing so rapidly that it is no longer possible to copy everything to the central site. So far, the solution is to keep "local" data at the decentralized sites, and "common" data at the central site. But this is not a valid solution for the future: Data is a common corporate resource, and timely access to it is getting ever more important.

In addition to the amount of local data increasing at a rapid rate, yet another issue is cropping up: More and more "local" applications also need access to central data files, so parts of central databases are frequently copied to the local sites. As soon as this data is updated centrally, the local data bases are no longer consistent with the contents of the central data base. Current DBMSs do not have any functions for managing this. Consequently, decentralized sites do not know whether the data they use is valid or not!

The interim "solution" to this problem, is to copy the databases regularly (e.g. once a week), or to collect all updates in a special file, which is transferred at regular intervals to all sites. Either of these methods is useful, but it does not completely solve the problem: It is only immediately after the file transfer that the databases are consistent; as soon as a single update is made to the central data base, consistency is lost.

Another problem is that all updates of such databases must take place centrally: If we allow updates at the local level as well as the central level, consistency is not only lost much faster, we can end up with updated information being replaced by "obsolete" data from the central database at update-time.

One approach to solving this kind of problem is to avoid centralization, and let every local site manage its own data. This means that for data which is really common to all sites, we end up with as many copies of it as we have sites. If the amount of common data is small, this may be acceptable, but in most cases it is not acceptable.

Yet another problem with this distributed approach of data management at each site is the problem of stuctural consistency between databases. Even in very distributed corporations, with high degree of local control of operations, there is a need for communicating information to higher levels of management, where this data is collected into corporate wide data. For this reason, as well as for reasons of accountability and controllability, the data structure used to store one type of data should be consistent across all local sites within the corporation. With complete local control, this will often not be the case.

Perhaps the main reason why HP3000 users are among those who need distributed databases the most, is that the HP3000 is so popular as a departmental computer. For this reason, a large number of the corporations that have HP3000s have more than one such computer. When you add to this the fact that the HP3000 has very good data communications facilities (to other HP3000s particularly, but also to other computers, e.g. IBM mainframes), you get a system that is almost "begging" to be used for the implementation of multi system databases.

III. CURRENT SYSTEMS

The currently existing multisystem databases (MSDBs) are all experimental. But these exipermental or pilot projects at least to some extent indicate the driection in which the world of distributed processing is moving. And even though a number of these projects are based in the academic community, which has not always been known for a commercial orientation, there seems to be a significant amount of realism behind many of the projects.

The selection of systems that we have done for this presentation is somewhat random or haphazard. But they should still represent a cross-section of the types of experimental multisystem databases currently under investigation at research institutions all over the world. Much of this presentation based on data presented at the Second and Third International Symposia on Distributed Databases, held in 1982 and 1984, respectively. The proceedings from these conferences are available in book form (see the chapter on references).

III. A: MULTIBASE

Multibase is a set of programs for accessing data stored in multiple databases. These databases may exist on identical or diverse computers. Multibase is intended as a commercial product, and as such is no longer a "prototype". Multibase is a product of Computer Corporation of America. It is currently only available for IBM mainframes.

There are four main reasons why we have included Multibase in this presentation:

1. Multibase is a "real" product, not just a prototype;
2. Multibase is a typical query-only multisystem database;
3. Multibase allows access to existing databases;
4. Multibase uses a copied catalogue concept.

This makes Multibase an "extreme" multisystem database from one point of view, and it makes it very easy to contrast Multibase with another - and very different - MDBS: POREL. We will have more to say about POREL later; for now let us concentrate on the features and functions of CCA's MultiBase.

Multibase is a product that allows relational queries to be made that access multiple databases, on multiple computer systems. Queries Multibase are formulated based on an integrated schema, which defines a "virtual" databases (called views) that each may consist of the data in relations stored on one or more of the computers in the network.

Some views may be "simple" in that they access only data stored in one relation, whereas others may be quite complex, and access data in multiple relations, with mapping of data from one field or item to another. Multibase also supports recalculation before integration. An example of this is when data from relation A specifies monetary values in Pesetas, and relation B specifies it in dollars. In the integrated view, such monetary values from relation A are recalculated as dollars before being introduced into the integrated view.

Another feature of Multibase is that it does not require the actual data storage to be relational. There are internal facilities in the system for relational retrieval of data from other types of databases, including hierarchical and network databases. This means that pre-existing databases of almost any type may be accessed from the query facility of Multibase.

A Multibase user has access to a language called DAPLEX for data definition and manipulation. This language is first used to set up the views, through definition of a Global Schema. Subsequent access to the integrated multi system database is though this schema. Additionally, DAPLEX Local Schemas are

defined for each actual database to be accessed. These schema map one-to-one to the Local Host schemas, which define each database in the "native" database definition language of the local computer system(s).

The overall organization of Multibase schemas is as illustrated below:

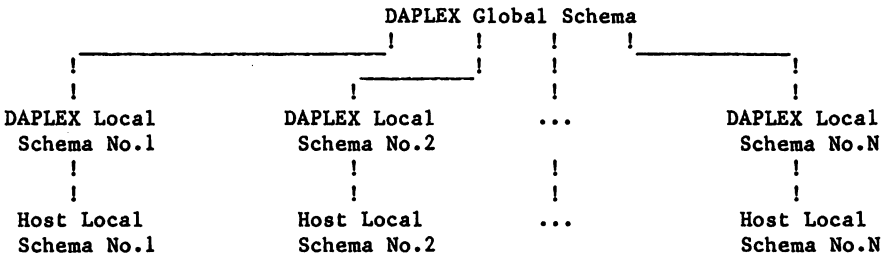

Figure 3.1: Multibase schemas.

When a user accesses Multibase, he does so through the Multibase Global Data Manager (GDM). The GDM finds out how to process the query from the Global Schema. The GDM splits up the query into a number of "single-site" queries i.e. queries that can be accessed with knowledge only of one DAPLEX Local Schema. The GDM then forwards these single-site queries, through the communications network, to Local Database Interfaces (LDIs), where the DAPLEX Local Schema is used in order to retrieve the necessary data from the actual database. LDIs then send their results back to the GDM, which performs any required coordination of data, or operations that require data from more than one host computer Finally, the GDM passes the results of the whole query back to the user.

We have illustrated the relationships between the GDM, LDIs, and schemas in the following figure:

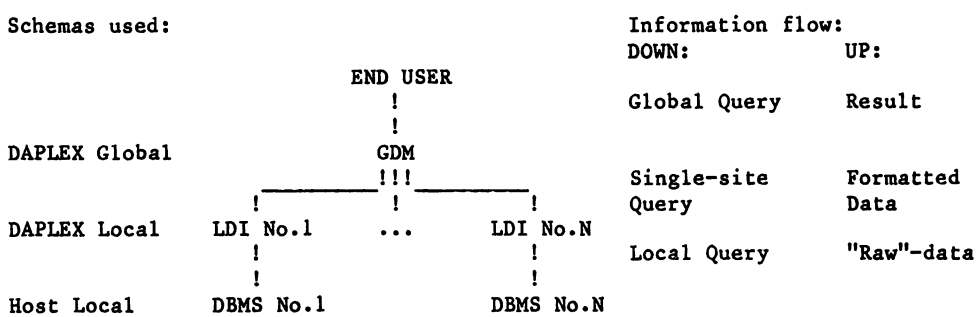

Figure 3.2: Multibase component interaction.

The Global Data Manager is the central piece of software in Multibase. It contains a number of advanced functions, including a description of each LDI concerning the capabilities - or lack of such - at each site within the network. This ensures that the queries sent to the LDIs only request functions that really are supported by the local Host DBMS. This function should help to keep the amount of work in creating Multibase LDIs for new systems low, which again means that Multibase might quickly become implemented on a number of diverse computers.

Among other GDM functions, which are not unique to Multibase, but rather standard for MSDBs, are global and local query optimization, and an auxiliary database for data needed to coordinate data from different local databases Multibase uses an internal DBMS to access this database in the final stage of preparing the results for the user.

The Multibase Local Database Interface (LDI), which gives DAPLEX support to each local host DBMS, is a relatively simple processor. Its main purpose is the translation of queries from the DAPLEX language to whatever the host supports. As has been noted previously, the GDM has knowledge of which functions are supported by each LDI/Host DBMS combination, which also simplifies the design of each LDI. For a typical DBMS, the effort necessary to implement an LDI is of the order of 3-6 manmonths.

We can probably assume that typical Multibase networks will contain two types of nodes: Those that support a GDM and LDI, and those that only support an LDI. This means that Multibase is not a uniformly distributed system, which may not matter much to actual users.

At the time of writing this paper, the author knew of no sites where Multibase has been implemented. But since the product is now commercially available the first real user experiences should be available soon.

III. B: POREL

POREL represents one of the other extremes of multisystem databases. POREL is a prototype developed at the University of Stuttgart in the Federal Republic of Germany. Among the major differences between POREL and Multibase are:

1. POREL is still only a prototype, not a product;
2. POREL only allows access to databases created with POREL;
3. POREL allows for all types of access, update and add as well as retrieval;
4. POREL uses a distributed calagoue, where some information may be retrieved from other sites;
5. POREL treats all users as "global".

In other words, we are looking at a database system with significant differences as compared to Multibase. Many of these differences, though on the positive side; they are **additional** features, not available in Multibase or comparable systems.

Perhaps the most negative difference, from a user's point of view, is that POREL requires all databases to be recreated. If data is to be copied over from an existing database, the user will have to develop the necessary programs on his own. This makes POREL primarily useful for dedicated, applications. This clearly contrasts with Multibase, which is primarily intended as an access method to existing databases.

Let us now take a closer look at POREL, which it definitely deserves: It is a very interesting system, and may point the way farther into the future then the much more restricted Multibase-system.

POREL is a distributed database management system developed for a network of interconnected minicomputers. POREL has been implemented on minicomputers with an 16-bit addressing range, and therefore consists - on each computer - as a set of interacting processes. All of POREL has been written in PASCAL. (This should make it possible to transport POREL to the HP3000 and HP1000 systems if desired; though I have not performed any evaluation of this possibility).

For computer-to-computer data communications, POREL uses X.25. A POREL system may be arbitrarily large, but the greater the number of processors, the higher the probability of failure in communications.

One of the most noteworthy features in the design of POREL is the emphasis placed on reliability and error-recovery. POREL is designed not only so that the failure of one node does not bring the network down, but the restart of nodes, including picking up where they were when they failed, has been carefully designed. (This may lead one to suppose that the computers used to implement POREL are unreliable, or that the level of programming performed is not reliable. There is no proof of either of these assumptions, though).

POREL transactions are described through either the use of a special query language (RDBL = Relational Data Base Language) or though special code available to PASCAL programmers (P-RDBL). This PASCAL-support is based on the programmer inserting special statements in his code, which is converted to procedure calls by a pre-compiler.

During the compilation of a PASCAL program with P-RDBL statements, or th compilation of a query stated using regular RDBL, a Network Independent Analysis is performed. Then, after all checks have passed at this level Network Oriented Analysis is performed. At this stage POREL may access nodes in order to retrieve information about data stored there.

When a complete transaction has been analyzed, it may be executed. This cause sub-queries to be sent to all relevant processors, and they are asked to retrieve the specified data. In the case of an update, a two-phase locking strategy is used, whereby first all data is locked (at all involved locations) and then the update is performed through the use of a COMMIT command. This allows for backing out of a partially completed update if one or more node(s) should fail during the update.

All POREL machines keep track of transactions in progress, and can therefore recover from the point where they were, should any kind of failure occur. The machines also keep track of which other systems are UP or DOWN, which helps avoid locking some data entries when it will not be possible to lock all the ones that are needed for the transaction, because one or more of the machines required are down.

POREL also has support for storing parts of a relation on different computers as well as for storing duplicate copies of a relation - or part of one - on separate computers. This will help keep the amount of data communications down during query processing, but it adds a lot of complexity to update processing. If we assume that query is the most common operation, and update is much less frequent, this may not necessarily impact performance of the overall system too much.

In managing multiple copies of a relation, POREL marks one as being primary,and the other ones as secondary. Updates of secondary copies may be delayed until the next time the data in this tuple of the relation is needed, which means that we can postpone some of the update activity until the system has "idle" time. POREL also keeps track of whether there are any outstanding updates to be performed on a secondary copy of a relation, to avoid letting users retrieved not fully updated information.

IV. HP3000 CAPABILITIES

The current capabilities of the HP3000 file and database systems do not include MSDB support. But there are ways of implementing some MSDB type functions. In this chapter we will look at some of these, both implementation of multisystem databases where only HP3000s are involved and where the HP3000 is a node in a non-HP network (in this case, an IBM mainframe oriented SNA-network).

IV.A: HP3000-TO-HP3000 MSDB'S

In order to access one HP3000 from another, the natural data communications product to use is DSN/DS. This product allows two or more HP3000s to be connected together, running either BSC protocol or X.25 over a public packet switched network (PSN). When using X.25 over a PSN, the HP3000 can also communicate with computers from other vendors that support this method of communication.

DSN/DS can also be used to communicate with HP1000 computers, both the BSC protocol and X.25 is supported on these computers. In this presentation we will concentrate on communication between two or more HP3000s all of which DSN/DS/X.25 over a packet switched network.

(HPs NS (Network Services) product, which handles communications over Local Area Network (LAN) can also be used for HP3000-to-HP3000 communication. For the presentation of DS in this paper, NS and DS may be viewed as providing essentially the same services. We will therefore only refer to DS, and it is up to the reader to supply this with a comment of "or NS" in each instance, if he so desires.)

DSN/DS allows a user on one HP3000 to log on to another HP3000, and to run those programs on the remote system which his account- and username give him access to. It is also possible to handle the logon etc. programmatically, so that the user is not aware of the connection. With DS/X.25, one physical line be used to access multiple computers simultaneously, by one or several users.

(X.25 allows up to 4095 simultaneous sessions across one link; no current HP3000 system can handle it, and the amount of data transmitted would be much higher capacities than is currently available: Typical DS/X.25 connections are at 9600 bps, or 1200 characters per second. Should 4095 users share this capacity, they could each send one character every 3.5 seconds!)

One of the many important features of DS, compared to similar offerings from other vendors, is the complete transparency of a DS-connection to software: A simple FILE-equation can be used to signal that a file resides on another computer; the file system will then automatically use Remote File Access (RFA) capabilitites to access the file on the remote system, in a fully transparent manner - as seen from the user and the application program.

The same kind of capability is available for databases, through DS Remot DataBase Access (RDBA). This provides the capability of accessing IMAGE databases on remote systems as if they were locally available.

RFA and RDBA are important capabilitites, but do not really support multi system database access. One of the reasons for this is that they do not allow for processing of queries on the remote computer before passing the result back to the host (or local) computer. For this reason, selecting data on other than an exact key match, will cause too much data to be transmitted over the data communications link.

Let me give a small example: Let us suppose we have an IMAGE master that contains information about customers. The key is probably the customernumber or the customer name. If we want to find the customers who are in the city Madrid (which could be none!), we will have to search serially through the set. In this case, if we use normal RDBA, information about about all the customers would be transmitted, and the application program would have to select the data it actually needs.

(There is only one available method for optimizing this access; that is to read only a few fields from the set, typically the key (customer-number) and the field we are scanning (city). For those that match, we can afterwards do a direct lookup by key to get all information about this customer.

In order to support true multi system databases, we will have to develop efficient programs that allow us to pass a query to the remote computer, instead of an IMAGE intrinsic call. For relatively simple types of queries, this should not be too difficult. The easy way to do it, is to develop inter-

process communication for the standard QUERY/3000 program, and use this as the vehicle for retrieving information from databases on multiple systems.

There is another facility that is needed as well. This is the ability to specify, in a DICTIONARY type of format, where the individual databases that form our overall multisystem database are located. This dictionary must be capable of defining different types of data transformations that should be performed on the local data before it is compared with data from other databases, i.e. before it is integrated into the overall multisystem database.

IV.B. HP3000-TO-IBM MSDB's

Many HP3000 users also have large IBM minframes as their central data processing systems. These users frequently communicate between their HP3000s and the maniframe(s), using either the BSC products (RJE, MRJE, or IMF), of the SNA products (NRJE or SNA IMF). For interactive communications, only two of these products may be used, IMF (for BSC) and SNA IMF. These two products are compatible from the point of view of a user program attempting to access data on the mainframe, since they have exactly the same intrinsics. There is only one slight difference in one of the intrinsics (OPEN3270, which is called only once by a program anyway).

There are three types of multisystem databases that can be created for a mix of HP3000s and IBM mainframes. These are:

1. HP3000 can access IBM, not the other way;
2. IBM can access HP3000, not the other way;
3. Access is allowed both ways.

For approach 1, above, programming need only be done on the HP3000. For access other way, it is necessary to implement programs on both types of computers even though approach 2 does not require access in both directions. This is because IMF is a "one-way" type of product, for HP-to-IBM access, and requires that communication is started from the HP3000.

Currently, there are some users who do interactive access to data stored on IBM mainframes from HP3000s. This is mostly done in a tailored manner where the application contains its own code for the actual data access on the mainframe. To the best of my knowledge, no general system exists. But a general approach has been described at a previous INTEREX conference, by the author of this paper. Below I will try to explain how multisystem database access to an IBM mainframe is possible with the current HP3000-to-IBM mainframe products: IMF and SNA IMF.

The lowest, user-accesible, level of both IMF products is a well-defined set of intrinsics. These intrinsics allow the user-program to emulate all funtions of an IBM327X terminal with 24 lines of 80 characters each (i.e. larger screen sizes and graphics are not supported). By calling these intrinsics, the user-program can access any IBM mainframe database accessible through a user-oriented program (e.g. a query faciliy).

The customized way these intrinsics are currently used is to allow program interactive access to specific programs, which again access data in one or more specific databases. By redesigning these programs, to work with a general query facility on the mainframe, and a data dictionary (which could be on either system) it is possible to allow the user to formulate a query on the HP3000, execute the query on the IBM mainframe, transfer the resulting data to the HP3000 via "screens", and finally reformat and display the data to the end-user. This kind of facility exists today on PCs, generally designed to work with a specific mainframe query-program, often from the same vendor as the PC

This kind of facility exists today on PCs, generally designed to work with a specific mainframe query-program, often from the same vendor as the PC software.

On the HP3000 we can take this approach one step further. We can develope a data dictionary facility on the HP3000 that allows for specification of mainframe **and** HP3000 databases (local and remote), and then allow the user to formulate queries based on all these databases. These queries must then be sp into subqueries, one for each computer, and transmitted to these system, where the database is located for execution.

For this approach to be really useful, it would be an advantage to providea vendor-independent access-method to the mainframe, since HP3000-users have different query-facilities on their IBM mainframes. This is a challenge to HP and to third parties who have HP3000-based query and data dictional products.

Another approach to multisystem databases in a mixed HP-IBM environment, is it implement an IBM mainframe oriented distributed database program, for example Multibase, on the HP3000. This approach would consist of two components: The necessary software for allowing the MDBS to access IMAGE databases (i.e. the LDI level of Multibase), and the actual software for managing distributed queries (i.e. the GDM facility of Multibase). This, of course, would have to be done by, or in cooperation with, the vendor of the MDBS.

Perhaps one of the first pieces of software that will really help us to move in this direction is HPSQL - an HP-developed, SQL-compatible query language the HP3000. This product could work as the access method to HP3000-based databases, and would make it easier to ensure compatibility between queries formulated based on HP3000-structured databases, and IBM mainframe database systems.

V. FUTURE TRENDS

In this paper so far, we have looked at the current "future of the art" and at most a few years into the future. Let me now look a little further ahead towards "the shape of things to come".

There are three specific trends that impact multisystem databases, and we will look at each of these in turn. They are all related to standardization, within the following areas:

i. Data communications facilities
ii. Database structures
iii. Database query languages

V.A. TRENDS IN DATA COMMUNICATIONS

There are three trends in data communications worth watching. They are:

i. Protocol standardization, which focuses on the Open System Interconnection (OSI) Model from ISO. The lower three levels of OSI are virtually identical to X.25. (IBMs SNA is another focus for standardization).
ii. Digitalization, which means that future communication services will be provided through all-digital packet- and circuit-switched networks; the modem is on its way out of basic system-to-system communications.
iii. Higher transmission rates, which makes the movement of significant amounts of data through public and private networks feasible without excessive re sponse times.

Taken together, these trends will lead to the availability of high-speed virtually error-free data communications between all points on the globe

(or into space, for that matter). In the 1990s the typical packed-switched interface will probably work at speeds up to 1 Mbit per second, rather than the current 9600 bps.

All significant vendors in the 1990 computer business will supply software and hardware to connect their systems to others, all supporting at least the four or five lowest levels of OSI. With standardized LAN facilities in the 10-100 Mbps range, local communication should also be standardized.

V.B. TRENDS IN DATABASE STRUCTURES

Perhaps the most significant trend in database structures of late, is the movement towards relational and semi-relationals structures. Another trend is the emergence and use of system-wide data dictionaries, that can be accessed from a multitude of programming languages, and that can perform additional tasks such as data base creation and restructuring.

Another trend in databases is the growing size - though not necessarily complexity - of databases in common use in businesses. This is a trend that has been made possible though the rapidly falling prices for direct-access storage media. The emergence now of optical-disk systems for archival datastorage, means that in the future "old" data will not be stored offline on magnetic tape (or paper, for that matter) in a vault somewhere, but will be accessible on-line at all times. This will severely impact database size.

Even on single computers, there is a trend towards integrating data from multiple databases into one common database - logically if not physically. This means that some of the basis for implementing distributed databases is already in place.

V.C. TRENDS IN QUERY LANGUAGES

Query languages of one form or another have existed since before the term "database" was invented. Most of the query languages in existence today (including HP QUERY/3000) are more or less ad hoc solutions that have developed slowly over the years, and have a set of functions dictated by combination of history and user demands.

With the trend towards a standardized set of database functions, i.e. the relational model for database structure, a standard for database query is emerging. This is the query language SQL (Structured Query Language, also known as SEQUEL), developed by IBM. Most vendors of relational database systems seem to have jumped on the SQL bandwagon, and sell an SQL-compatible query language together with their DBMS. This is now also true of HP.

More and more of the SQL-compatible query languages support a programmatic interface, so that database access from a program can take place through the same, simple interface that is used by on-line users. This is definitely an advantage, something anybody who has programmed using procedure calls for database access will recognize. This is even more true of programmers with experience from DBMSs such as IMS.

In the rapidly expanding world of multisystem databases, SQL can almost be considered the "de facto" standard for query languages. This trend will continue, and we will also see that SQL-compatible query languages will become available for non-relational (i.e. network or hierarchical) databases.

CONCLUSIONS

This paper has spanned a wide area of topics, and therefore has been unable to cover the subject-matter to any great depth. Additionally, the area of Multisystem Databases is a very large one, and it is growing rapidly. So we have only scratched the surface. But still, we may draw certain conclusions from the content of this paper. These are:

1. Multisystem Databases are possible;
2. Several approaches exist, with different capabilities;
3. The HP3000 datacommunications capabilities make it an interesting computer for implementation of multisystem databases;
4. Several future trends point towards making multisystem databases feasible in the near future.

So, in order to approach the year 2001 in a forward-looking manner, what do we do as HP computer users, need to think of? In this area, at least, we should consider the following:

1. Move towards relational or pseudo-relational database management systems, not just on HP3000's, but on all our computers;
2. Move towards standardized query/retrieval languages for databases, particularly systems such as HPSQL;
3. Use only internationally standardized data communiation facilities,e.g. X.25 for wide-area datacommunications, and IEEE 802.3 for local area networking.

By following the above guidelines, we should be on our way towards solving the data-processing and data-communications challenges of the next few decades.

REFERENCES

For a more complete description of Multibase, see: "An Overview of Multibase" by T. Landers and R.L. Rosenberg; in H. J. Schneider (ed): Distributed Data Base, North-Holland Publishing Company, Amsterdam, 1982.

This above paper - presented at "the second international symposium on distributed data bases" in Berlin, 1982 - gives a concise description of Multibase, it's capabilities and functions. For more information on this product, contact Computer Corporation of America (Ann Arbor, Michigan, USA) or their representatives.

For more information about POREL, see: "An Overview of the Architecture of the distributed Data Base System "POREL" by E.J. Neuhold and B. Walter; in the same book as quoted above.

For a more detailed paper on access to on-line IBM applications from an HP3000 using see: "Implementing Distributed Applicaitons in a Mixed HP-IBM Environment" by Rolf Frydenberg; in the proceedings of the HP3000 International Users Group conference, Montreal, 1983; available from INTEREX Santa Clara, California, USA.

Information about IMAGE Remote DataBase Access, and the products IMF and SNA IMF is available from your local HP sales office.

Other books that may be of interest are: James Martin: "Distributed Processing", Prentice-Hall, Princeton, NJ, USA, 1980. F.A. Schreiber and W. Litwin (eds): "Distributed Data Sharing Systems", North-Holland, Amsterdam, The Netherlands, 1984.

BIOGRAPHY

Rolf Frydenberg
has been with Hewlett-Packard Norge A/S for four months. His job is that of an Applications Engineer, specializing in Data Communicating and Programming Languages. Before joining HP last year, Rolf Frydenberg worked as an independent consultant and a software developer. He was one of the principals behind the HP-to-IBM data communications product VTS/IMAS. He has also worked on other types of data communications software.

Rolf Frydenberg has been a frequent speaker at INTEREX international and national conferences since 1982, mostly on topics related to data communications. Rolf Frydenberg has also written articles for trade magazines in the USA and Norway, and is the author of a book on selecting computer systems (in Norwegian).