Copyright © 1982 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 9, November 1982
Printed in U.S.A.

Database Systems:

A Real-time Database Management System for No. 5 ESS

By D. K. BARCLAY, E. R. BYRNE, and F. K. NG

(Manuscript received March 17, 1982)

A database management system was designed and implemented for the No. 5 Electronic Switching System. It is a specialized database management system based on the relational data model. The database is distributed and subject to stringent real-time constraints and reliability requirements. The system characteristics, capabilities, and design philosophy are described herein.

I. INTRODUCTION

The Bell System's newest addition to its family of electronic switching systems is the No. 5 ESS (Electronic Switching System).[1] It represents a departure from past local switching systems and it presented a unique opportunity to apply the principles of database management to a distributed switching network characterized by severe real-time constraints and stringent reliability requirements.

The No. 5 ESS provides a software-controlled switching capability for local telephone offices. The system consists of a central processor and a series of microprocessor-controlled modules that interface with the switching periphery. It is programmed in the programming language "C."

The database is based on the relational data model and distributed across each processor in the network. All accesses to data, both local and remote, are controlled by the Data Base Manager (DBM).

The modular hardware and software design allows offices to be economically grown. In addition, the introduction of a processor-independent language along with the distributed nature of the system will help ensure that the No. 5 ESS will keep pace with today's rapid technological advances, and the current trend of increasing demand for new feature development.

The database management system was designed to promote the aforementioned objectives of the No. 5 ESS. Current advances in database technology have been applied to this system to provide a more secure, more maintainable database that meets the real-time needs of an ESS.

II. THE ENVIRONMENT

2.1 Hardware architecture

The No. 5 ESS is a digital time-division switching system that will serve offices ranging from 1000 to 100,000 lines.[2] The minimum configuration consists of a central processor and a single interface module. Additional interface modules are added as needed to achieve additional capacity (Fig. 1). Each interface module can serve approximately 2000 lines. The No. 5 ESS capacity is therefore directly proportional to the number of interface modules.

The central processor serves as the hub of a star network. The central processor performs all functions that require a global view of the system. It serves as a resource allocator for global resources including the allocation of time slots used for intermodule communication. The central processor also serves as a central site for interfacing with telephone company personnel and any external operation support systems.

The tasks required to process calls are distributed to the interface

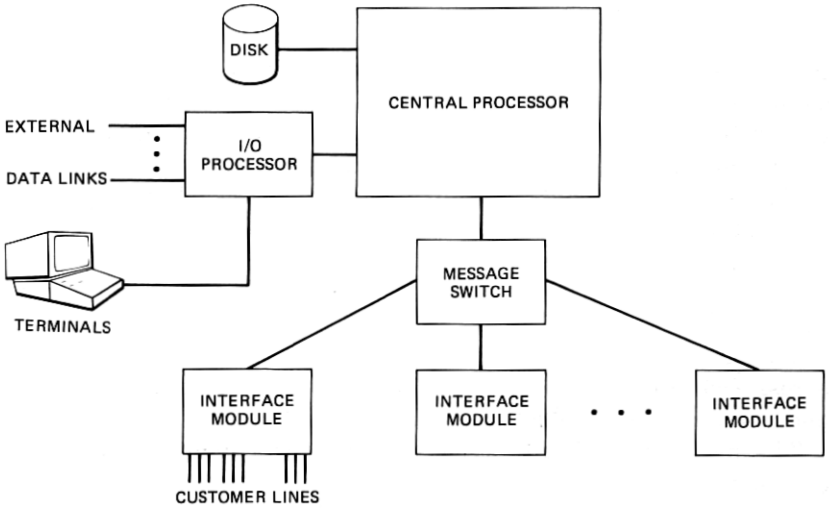

Fig. 1—Hardware architecture of No. 5 ESS.

modules whenever possible. This prevents the central processor from becoming a bottleneck in the system as the call capacity increases. A variety of peripheral switching equipment is connected to the interface module. Connections must be provided for customer telephone lines, digital connections between offices, and various service circuits for tones and announcements. All data reflecting the current configuration of the office reside in the database.

2.2 Software architecture

The software components for the No. 5 ESS include call processing, database management, administration, and maintenance.[3] All of these subsystems are supported by an operating system (Fig. 2). The operating system and the database manager reside in the central processor and each interface module. This fact, coupled with a single development language, provides a hospitable development environment that shields application programs from a multitude of processor dependencies. Software in the No. 5 ESS falls into two categories, processes and primitives. Processes are created and controlled by the operating system. Each process consists of a process stack dedicated to an instance of a process and reentrant code that can be shared by a number of concurrently executing processes. Each call spawns two processes. An origination process controls one end of the call, and a termination process controls the other end.

Primitives are globally accessible functions that control the use of a

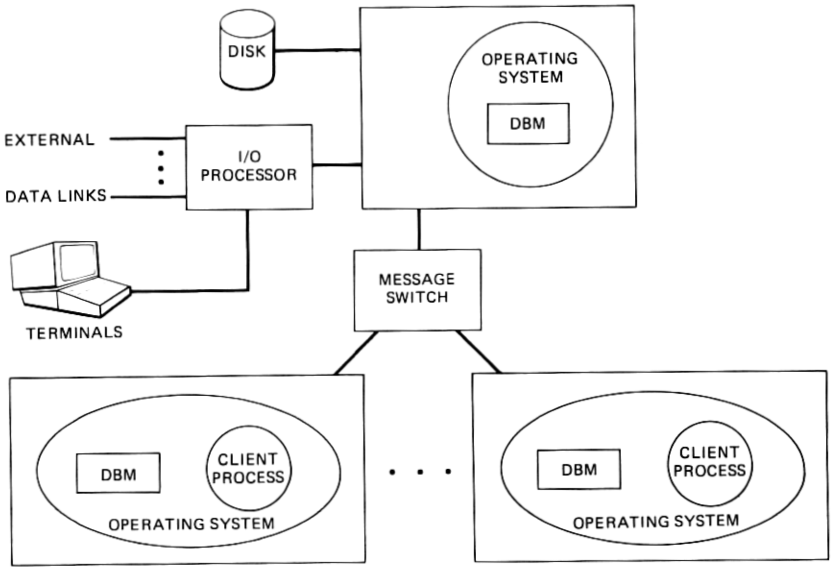

Fig. 2—Software architecture of No. 5 ESS.

system resource. The DBM contains a collection of primitives through which access to the database is controlled. These primitives are accessible to all processes in the system. They control concurrent database accesses within a single processor and isolate client programs from physical implementation details. This is the subject of Section IV. There also exists a process in each processor that is dedicated to database management. This process controls database communication across processor boundaries.

2.3 System characteristics

The No. 5 ESS database is based on the relational data model. The database is viewed as a collection of base relations. Base relations are permanently stored in the database. The No. 5 ESS DBM is a specialized Database Management System (DBMS) designed for electronic switching. Data accesses are performed via a procedural data-manipulation language. Certain characteristics of the system that have significantly influenced the design of the DBM are described below.

2.3.1 Real-time response

The primary function of an ESS is to process telephone calls. The requirement on data access time is rather stringent to enable the software to drive the telephone peripheries. A tuple of a relation must, on the average, be retrieved in the order of a millisecond.

2.3.2 Key access

Each piece of equipment in an office is referenced by its unique name in the processing of a call. Hence, the most frequent data access is based on primary key.

2.3.3 Stable schema

The introduction of new features to a switch is coupled with a new software release. The schema of the database remains unchanged within a given release.

2.3.4 Concurrent access

The database is a shared resource in the No. 5 ESS. Concurrent processes are active at any given time to access the database. Consequently there is a need to control concurrent access to the database.

2.3.5 Retrieval-oriented access

The main user of the database, call processing, is essentially a retrieve-only user. The data is changed by the telephone company personnel whose activity is less frequent and less real-time critical than call processing.

2.3.6 *Data integrity*

All ESSs in the Bell System have very stringent reliability requirements. It is essential for the DBM to maintain data integrity in order to meet such requirements.

2.3.7 *Predictable queries*

The query facility is used by the telephone company to update and verify the data stored in the switch. Traditionally, such a query interface is form-based and is unchanged during a software release.

2.3.8 *Non-casual users*

Only the application programmers who design the operational software of the No. 5 ESS directly interface with the DBM. These are sophisticated users who do not need a high-level interface to use the DBM. (Other users, e.g., telephone company personnel, interface with software built on top of the DBM.)

2.3.9 *Distributed system*

Currently, the No. 5 ESS is a distributed system. Its database manager and its data are distributed in the central processor and each of the interface modules.

III. DBM ARCHITECTURE

To achieve the access time necessary for a real-time system, a large portion of the No. 5 ESS database permanently resides in primary memory. Furthermore, call processing programs need to minimize database accesses. Early in the development cycle an attempt was made to design the database in normal form. This effort was abandoned because normalization tends to increase the number of database relations. This, in turn, increases the number of data accesses a program must make to retrieve the desired data.

Currently, the No. 5 ESS database of a single module office consists of approximately 200 relations. About 90 relations reside in the interface module and occupy 200 kilobytes of memory. The central processor database occupies 500 kbytes of memory.

3.1 *Data Definition*

The purpose of any database management system is to provide a measure of data independence. In the No. 5 ESS data definition is done off-line and changes can be introduced by redefining database relations and recompiling user programs. The goal is to introduce database changes with a minimum of client program changes. Because the schema remains stable during a given software release, it is not necessary that database definition be accomplished on line. A data-

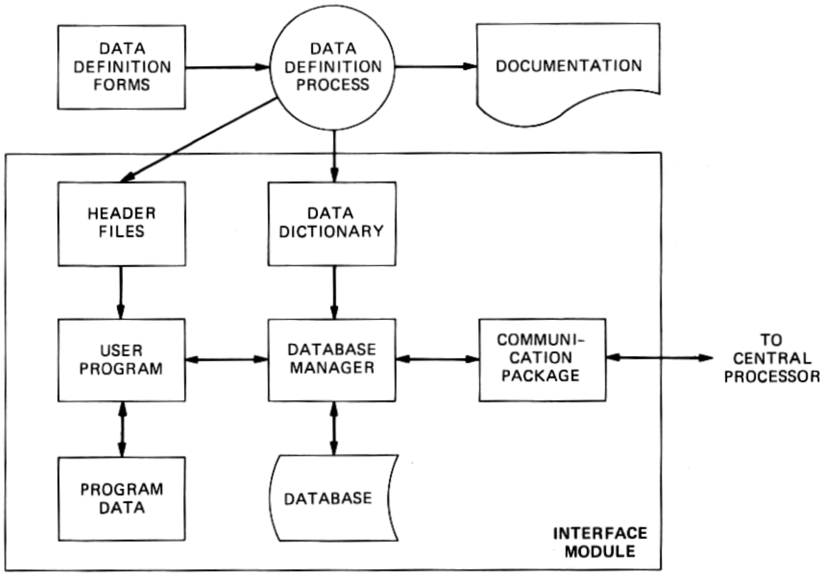

Fig. 3—Overall DBM architecture.

definition language accepts user definitions of the relations and their domains and generates user header files, data dictionaries, and documentation. Figure 3 shows the data definition process and the major software components in an interface module. The central processor contains similar components. The user header files contain the C-structure layout (or template) of the relation and are used by the users to compile with their programs. The data dictionaries describe all the relations, attributes, and domains in the database and are used internally by the DBM software.

3.2 Data manipulation

The No. 5 ESS DBM provides two levels of interface in accessing the database. The two levels of interface achieve different degrees of data independence for users with different performance and data-usage requirements. The basic interface is the tuple-level access that is used by call-processing programs. The user at this level can retrieve tuples of the base relations. The relations are designed from an operational point of view and are used by all No. 5 ESS internal subsystems. A higher level of interface, called the view-level access, is designed to provide a higher degree of data independence for users with less stringent real-time requirements. The view level access operates on view relations that are virtual relations formed by combining information from one or more base relations. The view level access provides an interface to the Bell Operating Company (BOC) personnel in a No. 5 ESS office, and also to an operation-support system through an external data link. Essentially, the view-level access supports a view of the No. 5 ESS data seen by the BOC personnel. Since the view-level access is used to interface with the external users, it is only available in the central processor. Figure 4 depicts the database management interfaces in the No. 5 ESS.

IV. DBM INTERNALS

4.1 Tuple-level access module

4.1.1 Concepts and capabilities

4.1.1.1 Transaction management. A transaction is a user-defined atomic unit of work consisting of one or more function calls to the DBM. Transaction management is required in the DBM for the following reasons.

(*i*) A transaction is used to ensure data consistency in a concurrent access environment. Within a transaction, the user sees a consistent

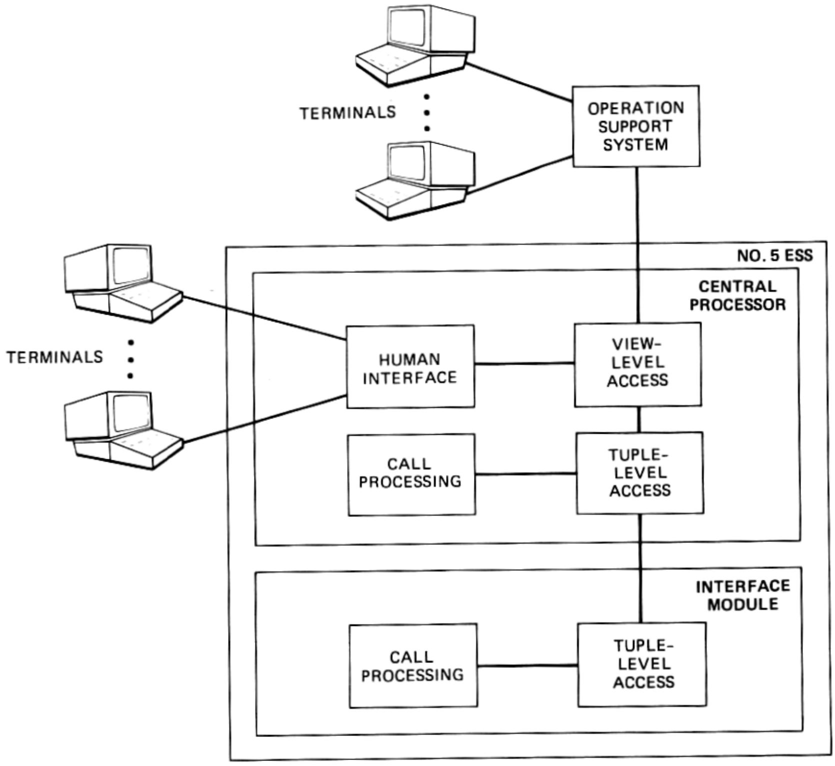

Fig. 4—DBM interfaces.

view of the database. It appears that the part of the database the user is processing is not affected by any update operation that may actually be introduced to the database during such period.

(*ii*) A transaction is used to ensure the integrity of data when updates are introduced to the database. Some of the integrity assertions on the data cannot be enforced when the data is being updated. These are the global integrity constraints that can only be enforced when all updates are completed. A transaction is used to defer the actual commitment of the updates to the database until all integrity checks are performed. The commitment of updates will take place at the end of a transaction. If the DBM had no concept of a transaction unit, it would be impossible to delay the enforcement of these constraints until the update was completed.

4.1.1.2 Levels of consistency. Three levels of consistency can be categorized in the concurrent access of data. The lowest level has no consistency at all. The user may read invalid data. Such a case happens when a user is reading some data while another user is writing on the data. The second level of consistency ensures that the user will always receive valid data but the data may not be consistent with other data in the database. The highest level of consistency ensures that the user will retrieve consistent data. This level of consistency is provided by the DBM.

4.1.1.3 Single command transaction. The transaction concept is intended to implement the highest level of data consistency. Some overhead is required to initiate a transaction and terminate a transaction. A simplification is made to the transaction concept. The scheme is called single-command transaction. Basically, a special set of tuple-level commands are designed to be used outside a normal transaction. Each such command represents an atomic unit of work on the database. It has the same effect as executing the counterpart of that command in a transaction. But, the overhead of initiating and terminating a transaction is largely reduced.

4.1.1.4 Tuple pointer. The use of a tuple pointer (a cursor on a relation) is required in the tuple-level interface for two reasons. One is performance related. It allows the user to access the tuple without going through the normal key-to-address transformation. This is particularly useful in an update operation when the tuple is located previously by a key-to-address transformation and an update is then applied to the tuple. Instead of going through the key-to-address transformation again, the tuple pointer could be used to readdress the tuple in a quicker manner. Second, a tuple pointer is essential for sequential traversal of a relation. A sequential traversal of a relation is generally based on the "next" operation. A tuple pointer, in this case, is used to identify the current position of the sequential traversal, which is used as the point of reference for the "next" operation. For the purpose of data independence, the tuple pointer presented to the user is only a logical entity that allows the user to reference a current tuple position.

4.1.2 Tuple-level interface

All the application programs resident in the No. 5 ESS access the data, regardless of its location, through the database manager at the local site via a set of interface primitives. These primitives are function calls as opposed to messages, which are the communication mechanism between processes. For accessing the data at a remote site, the DBM automatically generates all the necessary interprocessor queries (in terms of messages).

4.1.2.1 Basic design guidelines.

(*i*) Enhance efficiency without sacrificing data independence. The ESS database is isolated from application programs. A retrieval command causes some data to be copied from the database to the user space instead of providing a physical pointer to the database. Yet, to fulfill the real-time sensitive requirement of call processing, the capabilities of the interface primitives are somewhat restricted. First, only base relations are involved. Second, each primitive operates on a single tuple. Third, primary keys are normally used to identify tuples.

(*ii*) Provide a basic set of primitives. The DBM must provide a basic set of primitives that will serve all applications. The set of primitives should be "complete" so that the user can access any part of the database, but yet, it should be "basic" enough so that the DBM does not have to perform any part of the application for the user. Any special primitives that depend on an application should be provided by the individual user.

4.1.2.2 Tuple-level primitives. In general, there are four types of primitives. The first type is the administrative primitive, which handles the initiation and termination of a transaction and the opening and closing of relations. The second type is the tuple-retrieval primitive, which is executed inside a transaction. A tuple-retrieval primitive retrieves a tuple into the user work area and establishes a current position on the tuple retrieved (read-tuple, read-first-tuple, read-next-tuple). The third type is the tuple-manipulation primitive, which is executed inside a transaction. A tuple-manipulation primitive performs update operations on the tuple based on the implicit tuple pointer (insert-tuple, update-tuple, delete-tuple). The fourth type is the single-command transaction. This kind of primitive is executed as a transaction. It consists of retrieval and manipulation operations that are all based on primary key value (read, update, delete and insert tuple).

4.1.3 Layered software

The tuple-level access provides control and protection functions in addition to data access. It manages transactions, controls concurrent access and handles data distribution. It is supported by two layers of software: the access methods and the Software Demand Paging (SDP) package. The modular design allows the isolation of essential functions and thus reduces the impact of code changes to each development unit. The access methods provide data access functions by mapping tuple keys to logical addresses. Currently, three access methods are implemented: indexed, hash, and sequential. The SDP package provides the mapping of logical address to physical address and it also provides buffer management when the data is stored on a disk file.

4.2 View-level access module

The view-level access operates on view-relation tuples that are formed by combining information from one or more base relations. The view-level access is used to provide an interface to the personnel in a No. 5 ESS office. There are basically two types of view supported in the No. 5 ESS: the recent-change views and the verify views. A recent-change view is a view that is used by the BOC personnel to update the office database. Such a view must be updatable. A verify view is a view that is used by the BOC personnel to query the office database. A verify view is only required to be read-only. A recent-change (updatable) view can also be used to verify the data. However, the converse is not always true: there are verify views that may not be updatable.

4.2.1 View definition criteria

Not every view is updatable. Furthermore, a view, if defined arbitrarily, may even misrepresent the data relationships (i.e., produce-related data that is, in fact, unrelated according to the semantics of the base relations).[4] So there should be certain criteria in the definition of view. In general, there are more restrictions on a recent-change view than on a verify view. The reason is that a recent-change view, unlike a verify view, implies the existence of a mapping from view to base relations.

Two criteria for the definition of views for No. 5 ESS are observed.

(*i*) All views (i.e., recent-change or verify) must be *consistent* with the base relations.

(*ii*) All recent-change views must be *updatable.*

Some of the results developed in Ref. 4 are used as a guideline to define views.

To satisfy the consistency criteria, a necessary and sufficient condition exists. A view is consistent if and only if all joins, if any, performed to derive the view are lossless. A join is said to be lossless if the joining attributes of the two relations contain the key attributes of either one (or both) of the relations.

To satisfy the updatability criteria, more restrictions need to be imposed. A view is updatable if all the key attributes in the base relations used to derive the view appear in the view. The rule is derived from the assumption that all base relations used to define the view are involved in some kind of updates. However, this is not always the case in the views defined for No. 5 ESS. In a No. 5 ESS view, not all base relations used in defining the view are involved in some kind of updates. Some of the base relations are used purely for retrieval purposes even in an update view. These relations are mostly used to transform the external name, which is known to the BOC personnel, to one or more internal names, which are used by the system. Hence, the key attributes of some of the base relations will never appear in the view. Instead, a transformation is performed to map the internal key attributes to external key attributes. Hence, the conditions for updatability need to be revised as follows. A view is updatable if all the key attributes in the base relations used to derive the view appear in the view or there exists a one-to-one mapping between the key attributes in the view and the key attributes in the base relations that are actually involved in the update. Even though this is only a sufficient condition and not a necessary condition for updatable views, it is employed as a guideline to define recent-change views.

4.2.2 View definition

View definition is the process of specifying the attributes in a view and the mapping between the view and the base relations. The view-level access is responsible for performing view mapping from views to base relations and vice versa. The views handled by the view-level access are predefined views that can only be altered per generic. The views include all recent-change views and verify views defined for the generic.

The semantics of each view are defined by a customized mapping program. There are several reasons for not using a high-level relational language to specify views. First, according to the updatability criterion mentioned above, all base-relation tuples in a view are addressed by key values. The tuple-level access, which is designed for key-retrieval access, is much simpler and more efficient to use to perform these updates. Second, a high-level relational language is normally used to define mapping in one direction, i.e., from base relations to view. It cannot be used to define mappings explicitly from view to base relations.

4.2.3 View-level interface

The view-level access interfaces with the user through a set of view primitives that are similar to the tuple-level interface. The view-level access supports transactions on views. The basic set of view operations consists of retrieval, insertion, deletion, and update of individual view tuples using primary key. An auxiliary set, which is built on top of the basic set, consists of selection and count of view tuples based on a Boolean expression on the attributes of the view.

4.3 Special topics

4.3.1 Concurrency control

Concurrency control in the DBM is a mechanism that allows multiple users to access the data and protects them from getting inconsistent data due to concurrent updates. The policy on concurrency control has a great effect on the design of the concurrency mechanism. It determines the efficiency of shared data usage or, in other words, the degree of concurrent access.

Call processing is primarily a reader of the database and it contributes a large portion of the data-access activity in the No. 5 ESS. Since call processing requires real-time response, its access to the data should have high priority. Most changes to the database are not real-time critical and the frequency of such activities is rather low in comparison to call processing. Hence, our policy towards concurrency control is to provide efficient access to read-only users and perhaps less efficient access to read-write users.

The concept employed to control concurrent usage is based on data duplication.[5] All updates are performed on a copy of the real data. During the update period, all readers to the database can still access the part of real data that is being updated. When the update is completed, the updated copy will become the real data. The old copy will be discarded when all current read transactions on this relation are completed.

The copy scheme can present a problem when multiple users are updating the same part of the database. In fact, this mechanism can only be implemented effectively when no more than one writer to the database is allowed in the system to update the same relation. This, however, does not restrict the number of readers who have read-only access to the relation. Furthermore, it does not restrict the number of writers who are operating on the disjoint parts of the database. This situation fits quite well in the ESS environment where there are a lot of readers and relatively few and infrequent writers.

Since the concurrency control scheme is based on duplication of data, it is desirable to organize the storage structure in such a way that the amount of duplication is minimized. A tree-like storage structure is most suitable for this kind of application. The amount of data duplication required to perform an update in a relation consists of all the blocks along the path from the root node to the leaf node. All relations are in a tree-like structure regardless of what kind of access method is employed.

4.3.2 Distributed data access

A design goal for the DBM is to make the distribution of data transparent to its user so that the user need not be concerned with the actual site of where the data is stored. However, it does not imply that the DBM would be designed to handle the distribution of data in a general manner. Hence, the object is to define a data-distribution environment for the DBM that is more tailored to the needs of the No. 5 ESS operations.

There are two aspects that need to be examined in defining the scope of data distribution for the DBM. First is the type of data distribution that is actually found in the central processor (CP) and the interface modules (IMS). For example, a relation may be stored entirely in the CP, or entirely in an IM, or partitioned among the IMS, etc. Second, there is a need to determine the kinds of data interaction between the CP and the IMS. For example, it could be that a process in the CP may request data from the IM, or a process in the IM may request data from another IM, etc.

4.3.2.1 Types of data distribution. The complexity of data distribution depends on the granularity of the unit of data distribution. A unit of data distribution is the set of tuples in a relation that will always be physically stored in a site. A simple way to define a unit of distribution is to use the entire relation. In other words, the entire relation must be stored in one site. If multiple sites have the same relation, it will be fully duplicated. In this case the granularity of the unit of data distribution is the largest. Smaller granularity can be formed by partitioning the relation. In general, a relation can be partitioned horizontally, vertically, or both ways. A horizontal partition can be defined by a Boolean expression on the values of a set of attributes. A vertical partition can be defined by a projection on a set of attributes. A combination of horizontal and vertical partition is possible. In viewing the No. 5 ESS data, the partitioning is more restricted than those described above. First, there is no vertical partitioning. Second, horizontal partitioning exists, but it is rather restricted. No general Boolean expression is used in defining the partition. The relation is partitioned according to whether the tuples are used in a particular IM. Third, some relations are not partitioned at all.

The types of data distribution in the first issue of No. 5 ESS can now be categorized as follows.

(*i*) CP single. The relation is stored entirely in the CP. No data is stored in any IM.

(*ii*) CP-IM redundant. The entire relation is stored in the CP and is duplicated in each of the IMs.

(*iii*) IM reductant. The entire relation is duplicated in each IM. No data is stored in the CP.

(*iv*) IM partitioned. The tuples of a relation are partitioned in the manner described above in each of the IMs. No data is stored in the CP.

4.3.2.2 Data interaction between sites. The complexity of the DBM software for distribution depends on the data interaction between the processes. It is necessary to examine the kinds of data interaction that exist between the CP and the IMs, and among the IMs. Currently, most call-processing programs access data in its resident processor. Hence, most data access is done at the local site. Since most global data updates are from the central processor, these global updates can be controlled in a single site. Hence, the central processor naturally becomes the primary site for locking. Local updates are handled individually at each local site. The control of all global updates at the central processor has significantly simplified the mechanism for global concurrency control.

Based on the data usage of existing subsystems, three variations of data request can be identified. They are CP local, IM local and CP-IM remote. CP local means that the data request is originated from the CP and the data is located in the CP. IM local means that the data request is originated from the IM and the data is located in the IM. CP-IM remote means that the data request is originated from the CP and the data is located in the IM. CP local and IM local are mainly used by call-processing programs that access data in the local processor. The CP-IM remote is used by the recent-change and verify system that performs administration functions on the database.

V. SUMMARY

The database management system (DBMS) is designed and implemented. It is a real-time database management system based on the relational data model. The DBMS provides two levels of interface in accessing the database. The two levels of interface achieve different degrees of data independence for users with different performance and data-usage requirements. A data-definition facility is available for the user to define the schema of the database. The data-definition process is done off-line to reduce software complexity in the No. 5 ESS. The DBMS allows multiple users to access the database concurrently. To satisfy real-time requirements, the concurrency control mechanism is devised to provide efficient access to the readers and somewhat less efficient access to the writers of the database. Since most global updates are initiated in the central processor, it becomes the primary site for locking. The control of all global updates at the central processor has significantly simplified the mechanism for concurrency control in a distributed environment.

REFERENCES

1. W. B. Smith and F. T. Andrews, Jr., "No. 5 ESS—Overview," Bell Telephone Laboratories, U.S.A., International Switching Symposium, Montreal, Canada, September 1981.
2. J. H. Davis, J. Janik, Jr., R. D. Royer, and B. J. Yokelson, "No. 5 ESS—System Architecture," Bell Telephone Laboratories, U.S.A., International Switching Symposium, Montreal, Canada, September 1981.
3. S. M. Bauman, R. J. Carline, J. S. Nowak, and H. Oehring, "No. 5 ESS Software Design," Bell Telephone Laboratories, U.S.A., International Switching Symposium, Montreal, Canada, September 1981.
4. A. K. Arora and C. P. Carlson, "The Updatability of Relational Views Based on Functional Dependencies," Proceedings of Computer Software and Application Conference, Chicago, Illinois, November 1979, pp. 415–420.
5. Y. E. Lien and P. J. Weinberger, "Consistency, Concurrency and Crash Recovery," Proc. ACM SIGMOD Int. Conf. on Management of Data, Austin, Texas, 1978.