IEEE JOURNAL ON

SELECTED AREAS IN COMMUNICATIONS

DECEMBER 1993 VOLUME 11 NUMBER 9 ISACEM (ISSN 0733-8716)

A PUBLICATION OF THE IEEE COMMUNICATIONS SOCIETY

..... J. R. Haritsa, M. O. Ball, N. Roussopoulos, A. Datta, and J. S. Baras 1360

MANDATE: MAnaging Networks Using DAtabase TEchnology

Jayant R. Haritsa, Michael O. Ball, Nicholas Roussopoulos, Anindya Datta, and John S. Baras

Abstract—**In recent years, there has been a growing demand for the development of tools to manage enterprise communication networks. A management information database is the heart of a network management system—it provides the interface between all functions of the network management system and, therefore, has to provide sophisticated functionality allied with high performance. In this paper, we introduce the design of MANDATE (MAnaging Networks using DAtabase TEchnology), a proposed database system for effectively supporting the management of large enterprise networks. The MANDATE design makes a conscious attempt to take advantage of the special characteristics of network data and transactions, and of recent advances in database technology, to efficiently derive some of the required management functionality.**

I. INTRODUCTION

IN today's global marketplace, most large-scale enterprises have widely dispersed manufacturing and commercial operations for both economic and political reasons. For example, General Motors has manufacturing plants spread over the United States, Europe, and Japan. In order to effectively coordinate the functioning of a distributed enterprise, the subsidiary units need to be connected by a communications network. As the enterprise grows in size, its communications requirements increase correspondingly. The enterprise networks of the future are projected to be large agglomerations of subnetworks such as LAN's (local area networks), MAN's (metropolitan area networks), and WAN's (wide area networks). These enterprise networks are expected to be heterogeneous in several dimensions: First, the underlying physical transmission facilities may be "mixed media." For example, a local area network in Baltimore built with copper cables may be connected to a wide area network covering the Eastern United States based on fiber-optic technology, which is linked to the European communications system by satellite. Second, different subnetworks may be purchased from different vendors due to economic, performance, or historical reasons. For example, a company that uses SNA networking technology supplied by IBM may take over a company that has AppleTalk as its internal communication mechanism. Therefore, individual subnetworks may have different vendor-specific network management systems. Third, the information being transmitted over the network may be "multimedia," that is, semantic differences exist in the types of transmitted information. For example, video images may be transmitted on the same channels as those carrying telephone calls. Finally, individual users of the network may differ in their performance objectives. For example, users needing the network for data transfer may require high throughput while others, whose concern is voice communications, may require low call blocking probability.

For these reasons, future enterprise networks are expected to be highly complex in their transmission, performance, and communication characteristics. Due to this complexity and the disparity among management systems for individual subnetworks, efficient management of an enterprise network is an extremely challenging problem. Therefore, there is a clear need for research and development of network management tools.

Network researchers are in common agreement that a (conceptually) global network database, which contains all management-related data, is central to the development of an efficient network management system (e.g., [4], [34], [2], [30]). The database is required to store information on network and system configuration, current and historic performance, trouble logs, security codes, accounting information, etc. [18]. In OSI parlance, this database is called a Management Information Base (MIB). A practical example of an MIB-based architecture is DEC's EMA (Enterprise Management Architecture), where a Management Information Repository is defined as a central component of the Director [8]. While there has been intensive research on network management systems in recent years, comparatively little has been published with respect to the actual design and implementation of an MIB. In this paper, we introduce the design of MANDATE, a proposed MIB system for effectively supporting the management of large enterprise networks. We have attempted, in this design, to identify the basic mechanisms that need to be incorporated in network MIB's. Although this design has not been implemented in a network management setting, certain components have been implemented and tested in other contexts. In particular, the incremental client–server architecture we propose for use in MANDATE has been implemented and extensively tested [28], [27]. Moreover, experiments related to interfacing optimization algorithms with databases are given in [3]. As part of our future research agenda, we plan to test and tune the MANDATE design by

Manuscript received June 18,1992; revised March 16, 1993. This paper was presented in part at the 3rd International Symposium on Network Management, San Francisco, CA, April 1993.

The authors are with the Institute for Systems Research, University of Maryland, College Park, MD 20742.

IEEE Log Number 9212156.

implementing these mechanisms on a network testbed.

The guiding principle of the MANDATE design is to have the network operator(s) interact solely with the database; that is, from the operator's perspective, the database logically *embodies* the network. Whenever the operator wishes to make changes in the network functioning, such as changing the routing scheme, for example, the operator merely updates the appropriate variables in the database. The actual implementations of these changes in the physical network are made by the database system. This design approach allows the operator to concentrate on what has to be done, rather than on the mechanics of implementing the decisions. A second important aspect of the MANDATE design is that it is a bottom-up design, not a modified version of commercially available database systems. This results in a system architecture that is tailor-made specifically for network management. Finally, we have made an attempt to identify and take advantage of the special characteristics of network management data and transactions, and of recent advances in database technology, to efficiently derive the required MIB functionality.

The remainder of this paper is organized in the following fashion. In Section II, the role of database systems in network management is discussed in detail. In Section III, the related work on database support for network management is briefly reviewed. Then, in Sections IV and V, we describe the data and transaction modeling aspects of network management. In Section VI, we describe the design of MANDATE and how our design proposes to achieve some of the required MIB functionalities. Finally, in Section VII, we summarize the main conlcusions of the study and outline future research avenues.

II. Role of Databases

The ISO/ANSI standards committee [7] has classified the sophisticated functionality required of network management systems into the well-known six categories of configuration management, fault management, performance management, security management, accounting management, and directory management. The functional architecture defined by these six categories clearly identifies the different facets of network management and control, and enables a modular approach to be taken towards designing network management tools. However, there is considerable overlap and interaction between the various management subsystems. For example, fault management and performance management are closely interrelated, since poor performance is often the only visible symptom of a fault deep down in the system. Similarly, detecting a faulty resource and isolating it from the remainder of the network requires both fault management and configuration management. In order for the various management modules to coordinate their activities, a common "public workspace" or *database* is necessary. Therefore, a logically integrated database is the *heart* of a network management system [1]—it provides the interface between all functions of the network management system, as shown in Fig. 1. This database, or MIB, is the conceptual repository of all management-related information. The MIB defines the set of managed objects visible to a network management module, and the network operators use the MIB to communicate all commands to the physical components of the network.

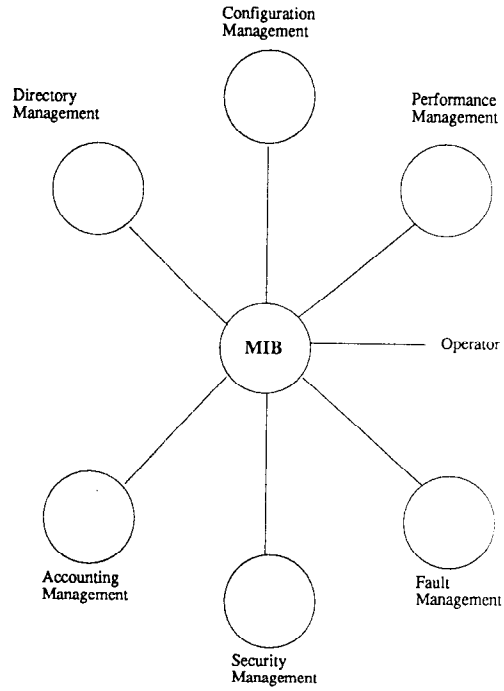


Fig. 1. Network management model.

A. Requirements on MIB

Ideally, the MIB module of an enterprise network management system should provide the following functionalities:

1. *Homogeneous Interface:* Present a uniform interface to the operator that is independent of the individual subnetwork characteristics.
2. *Graphical Interface:* Allow the operator to view the network at any level of detail; that is, to graphically navigate the MIB.
3. *Scalable Design:* Add new subnetworks or increase the functionality of existing subnetworks without requiring complete restructuring of the database.
4. *Fault Tolerance:* Operate 24 hours on-line since the MIB is the core of the network management system.
5. *Real-Time Response:* Store and process, in real time, the "network health" data which is continuously gathered by external network monitoring tools.
6. *Temporal Views:* Provide a "snapshot" of the network as of some real-world time instant. This is necessary for post-mortem fault and performance analysis.
7. *Active Mechanisms:* Support triggers that recognize and respond to special network situations (as reflected by the data) without requiring operator initiation.
8. *High-Performance:* Minimize the overhead of network management on the performance of the network. In addition, the network management performance should gracefully degrade under overload conditions.
9. *Decision Support:* Answer "what if" questions (for example, by executing on-line analysis algorithms), thus

helping the operator to evaluate the potential impacts of different control decisions.

10. *Embedded Control:* Efficiently execute on-line control algorithms (for example, expert systems) to adapt the network routing, configuration, etc. in response to changes in the network traffic or connectivity.

From this list, we see that an MIB has architectural requirements (fault tolerance, scalability, triggers), interface requirements (homogeneous, navigational), temporal requirements (real-time response, temporal views), automation requirements (active mechanisms, decision support, control), and performance goals. Clearly, the design of the MIB is key to providing all of these complex functionalities in an integrated fashion.

B. Need for New MIB Design

Since current database technology is fairly mature, one might think that using a popular database management package (e.g., ORACLE [22], INGRES [21]) should be sufficient for implementing a network management MIB. There are several reasons, however, why existing DBMS products are not satisfactory from the network management perspective:

- Standard off-the-shelf DBMS's lack many of the required MIB functionalities, such as real-time capabilities and decision support facilities.
- Conventional DBMS's have been developed for the commercial query processing environment and are primarily geared towards applications such as banking, where the focus is on naive human users interactively performing transactions. In network management, however, software programs control the network behavior with human intervention restricted to skilled operators.
- The objective of conventional DBMS's is to efficiently implement a transaction model that provides the so-called ACID property; that is atomicity, consistency, isolation, and durability [23]. However, this transaction model is unsuitable for processing of network management data, since these properties are not always essential here. For example, in banking databases, all updates have to be guaranteed for the database to be correct, and this guarantee is provided by conventional DBMS's. In the network environment, however, updates to "network health" information such as packet retransmission rates or node queue lengths are not "sacred"—failing to register updates has only performance implications, but certainly no correctness implications. An MIB can use this property of network data to derive some of its functionality. For example, under overload conditions, it can continue to provide real-time response to critical network monitors by selectively ignoring the updates of less important sensors.

In summary, the network management environment is a specialized application area with unique characteristics that can best be taken advantage of by a database system that is built specifically for this environment.

III. Related Work

While network management has been an important research topic for the past several years, comparatively little work has been done, however, with respect to the database management aspect of network control. In this section, we provide a brief review of these papers.

Issues similar to those addressed in this paper were considered in [1]. The focus in that work was on evaluating how conventional relational DBMS packages would serve in the role of an MIB, and suggesting network-related modifications to these conventional packages. In contrast, our focus is on developing a new DBMS whose design is tailor-made for network management.

The NETMATE project at Columbia University [11] has also investigated many of the issues discussed here. The project has primarily concentrated on the model of the network and its architectural relationship with management tools. In particular, a novel approach to network management is described in [35]. This paper presents a data-oriented approach to network management, in which network management functions are specified as data manipulation statements. The goal is to use the expressive power of database query languages to construct network management functions such as tests and alerts in an interactive, declarative, and set-oriented fashion. This approach appears promising and integrates nicely with the "data-centric" design described in this paper.

An overview of the issues involved in implementing the MIB interface definition laid down by the OSI Standards Committee is presented in [4]. The issues considered included the choice of data model, the architecture for distributing network management data, and the mechanisms for ensuring integrity of replicated data. While the paper describes several of the functionalities to be provided by an MIB, it does not, however, provide a detailed design for achieving these functionalities.

Data modeling and data location issues are also discussed in [24]. A combination of Entity-Relationship model and object-oriented techniques is used in their approach, and a functional architecture for providing network management services is outlined. In [19] and [10], techniques are proposed for modeling the relationship between logical OSI objects and their concrete realization in a real implementation.

A Layered Attributed Graph was proposed as a formal mechanism to model a network in [34]. Different graphs, each representing a single layer of a seven-layer OSI network, are set into a formal layering relationship resulting in a layered attributed graph. It was suggested that this mechanism could be used as the basis for the design of a network management DBMS. The practicality of this approach remains to be seen.

Very recently, several books that are devoted exclusively to network management have appeared (e.g., [30], [17], [2]). These books highlight the importance of database tools in developing network management systems, but focus more on the functionality requirements and evaluation of such tools and less so on the design aspect and mechanisms for realizing the functionality requirements.

Finally, there are numerous papers on expert systems for network management (see [13] for a detailed survey), all of which rely on an underlying knowledge base on which to base their inferences. These papers usually assume the existence of

a database (typically in the form of rules) and develop expert systems on top of this knowledge base.

IV. Network Data

The first step in designing a database system is to understand the properties (semantics) of the data items that are resident in the database and to understand the properties of the tasks (or transactions) that store, process, and retrieve this data. Network management data can be broadly classified into three types: sensor data, structural data, and control data, as shown in Fig. 2, which describes a high-level abstraction of the MIB data model. As explained later, the structural data describes the physical and logical construction of the network, the control data captures the operational settings of the network, and the sensor data represents the observed state of the network.

A. Sensor Data

The sensor (or measurement) data is the raw information that is received from the network monitoring processes, and includes variables such as node queue lengths, retransmission rates, link status, and call statistics. The sensor data provides the primary input for three of the six OSI network management categories: accounting management, performance management, and fault management. It represents the current "health" of the network in terms of the network's usage and operational quality. Typically, each sensor's data arrives at a regular frequency under normal network operation. However, under fault or overload conditions, sensors may generate data at a higher rate than normal. Another possibility is where a sensor supplies data only when an extraordinary event occurs (such as a link going down), or only upon explicit request from the MIB control processes. For example, in the Internet, trap-directed polling is employed for dealing with extraordinary network events [25]. Here, whenever an extraordinary event occurs, the managed network element sends a single trap to the MIB, and the MIB is then responsible for initiating further interactions with the network element. Since the traps are sent unreliably, the MIB also employs low-frequency polling of managed elements to determine their operational status.

Sensor data can be divided into two groups: persistent and perishable. The persistent data consists of sensor data, whose utility is long term, and therefore needs to be maintained permanently in the database. Critical data such as customer billing information, network alarms, and security violations belong to this category. Due to the requirement of permanence, persistent sensor data requires the complete set of recovery mechanisms (i.e., logging, mirroring, checkpointing [23]) similar to those provided by commercial

Perishable sensor data, on the other hand, is data that is of "limited time utility" in the sense that its current value is valid only until the network characteristic that is being monitored retains that value. Data such as node queue lengths, retransmission rates, and most other dynamic performance statistics fall into this category. There is no need for logging of these updates since the information will be out-of-date by the time the MIB recovers from a failure. Also, unlike the persistent sensor data, updates to perishable sensor data are not

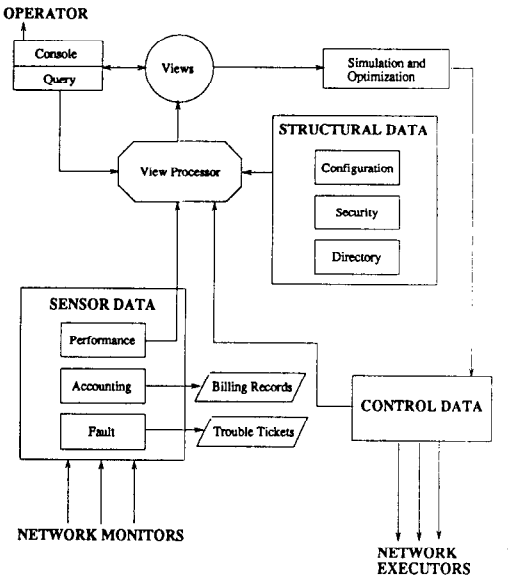


Fig. 2. Model of MIB.

"sacred" since every individual sample may not be essential; therefore, ignoring updates occasionally does not have serious implications. While the perishable sensor data has only limited time utility with respect to the immediate operation of the network, it may be necessary to retain a *history* of the data values for long-term postmortem performance and fault analysis. In order to fully implement this feature, a new version has to be created for each update of a perishable data item. From a practical storage perspective, however, it may be necessary to implement a coarser granularity of versioning, such that a new version is created only periodically (say, every tenth update) or only when the value of the observed variable has changed appreciably from its immediately previous archived value (say, by more than 10%). Further, it may be sufficient to version only those variables that are of critical importance in tracking the state of the database, such as the link utilizations and the number of retransmissions, and not verion less important variables such as the number of null-header packets or the byte count in individual packets.

In current large networks, the quantity of sensor data that is gathered may be as large as 20–30 gigabytes per day [6].

B. Structural Data

In contrast to sensor data, structural data is composed of "static" (slowly changing) network information such as the network topology, the configurations of the network switches and trunks, the data encryption keys, and the customer description records. This data provides the primary input for the remaining three OSI network management categories: configuration management, security management, and directory management. A point to note here is that, unlike sensor data, structural data is valid even when the network is not in operation.

Most of the structural data is stored at system initiation time. This data is changed at a moderate rate consistent with more classical database applications. For example, events

which would require structural data updates include adding a new switch to the system, adding a new customer, or having a breach of security. The structural data needs to be recoverable for monetary reasons (customer records are of vital importance), for efficiency reasons (restart quickly from a database crash), and for security reasons (accessing copies of data encryption keys remotely over the network could lead to security compromises).

In a typical large network, the quantity of configuration data depends on the level of detail at which the network equipment is represented, and may be of the order of several gigabytes.

C. Control Data

The final data category is control data, which captures the current setting of network tuning parameters such as the maximum flows on individual trunks, the traffic split ratios on the output links of switches, and the routing table. The process for changing an existing set of control settings is usually initiated by the network operators. Alternatively, the changes may be automatically triggered as a function of the information contained in the sensor data. For example, if there is a serious security violation (such as introduction of a virus) at a node, the links going through the node may be automatically shut down pending investigation of the problem by the network operators. In addition to the current parameter settings, the control database also stores a library of predefined control settings (often called "profiles") that reflect the appropriate settings for a variety of common traffic patterns and network configurations. For example, different suites of settings may be appropriate for day and night traffic.

In order to support the functionality requirement that operators should be able to obtain historical views of the network state, it is necessary to maintain a record of changes that are made to the structural and control data. However, since these changes occur at a moderate rate, we expect that the overhead of maintaining the update history will not be significant.

V. Network Transactions

Having discussed the characteristics of network management data, we now move on to considering the various types of transactions that operate on the data in the MIB. We note that we assume, in this paper, that transactions use a lock-based paradigm for accessing data objects, where shared locks are used for reading objects and exclusive locks are used for updating objects.

A. Sensor Data

Two distinct groups of transactions, "updates" and "readers," access the performance data (perishable sensor data). The updaters are network monitoring tools, while the readers are internal MIB processes. The updaters work in private data partitions since they update different sets of network variables and, therefore, do not interfere with each other. These updates are different from typical database updates in that the updated value is independent of the current value of the data object. Such updates are referred to as "blind writes" [5]. Since the performance data is versioned, readers can always read the data that they want without delay. Therefore, due to the absence of Read–Write and Write–Write conflicts, no explicit concurrency control is necessary for the performance data. If it is required to ensure *atomicity* of the set of updates made by each updater, updaters will have to follow a "degree 1" lock protocol [14] (update locks are held until the end of the transaction) while accessing data objects. However, since it should not be necessary for perishable sensor data to be recoverable, for performance reasons, we intend to allow updaters to follow the less restrictive "degree 0" lock protocol (update locks are short-term).

For accounting and fault information (the persistent sensor data), the network monitors append records to existing tables. The MIB internal processes may both read and update these persistent records. For example, a network monitor may register a trouble ticket in the fault database. Once the fault is fixed, the trouble ticket is updated by the operator to reflect this fact. Due to the concurrent reading and updating and the sensitive nature of the data contained in the accounting and fault tables, the MIB internal processes have to follow the classical "degree 3" locking protocol used in conventional database systems (both read and update locks are held until the end of the transaction) in accessing this data. However, the network monitors may only have to use a degree 1 lock protocol if their update accesses are restricted to appending new records and do not involve altering the contents of existing records.

B. Structural Data

Strucural data can be both read and written by the network operator(s) or by MIB control processes. For example, the addition of new equipment or facilities are usually entered into the MIB by the operator, while the configuration details of network switches and trunks may be automatically handled by MIB processes. Standard network management operations, such as inventory verification and customer authentication, are typically executed on the structural data. Since it is possible that multiple processes may access the same structural data simultaneously, structural data transactions have to follow the complete (degree 3) locking protocol in making their data

C. Control Data

Control data can be both read and written by the network operator(s) or by MIB control processes. The process for changing an existing set of control settings is usually initiated by the network operators. Alternatively, the changes may be automatically triggered as a function of the information contained in the sensor data. For example, if the operator observes from the sensor data that some links are becoming excessively utilized, he/she may decide to replace the routing scheme that is currently employed by a different scheme. Another source of change for the control data is that produced by reexecuting the optimization algorithms to reflect changes in the network configuration or activity profile, thus generating a new set of control settings.

Control parameters may be either under operator control or under automatic control. In the former case, the operator

manually determines the setting of the control parameter while, in the latter case, the MIB's internal processes automatically update the control settings. A facility will be provided in MANDATE, whereby a control setting may be moved from automatic control to manual control and vice versa. This allows the operator to assume full control under emergency or unanticipated situations. In a properly designed network, no more than one process should be able to update a given set of control variables at a time (it is meaningless to have concurrent updaters since the control of the network then becomes a function of the order in which the transactions are processed.) Therefore, concurrency control is not required for regulating access to control data. However, the transaction construct is necessary for installing the updates in order to ensure the atomicity of the updates (half-implemented control settings may cause havoc in the network). Changes to the control data trigger *network executors* (see Fig. 2), which are processes that actually implement in the physical network the control structure that is logically described by the updated control data.

D. Subnetwork Managers

In the most general setting, some of the objects managed by the MIB will include higher-level entities that have their own local network manager [25]. These subsidiary network managers may request information from the MIB in order to perform their local network management, or may inform the MIB of significant updates to their configuration or status.

VI. The MANDATE Design

In the previous sections, we identified the special characteristics of network management data and transactions. We follow up in this section by describing how the MANDATE design proposes to use these special characteristics, and recent advances in database technology, to achieve some of the required MIB functionalities.

A. Data Processing Model

As mentioned in the Introduction, the guiding principle of the MANDATE design is to have the network operator(s) interact solely with the MIB; that is, from the operator's perspective, the MIB embodies the network. Therefore, whenever changes have to be made to the network topology, routing scheme, switch software, etc., the operator merely initiates that update the corresponding data objects in the MIB. The actual implementation of these changes in the physical network will be made by execution processes that are activated or triggered by the database system. The design approach results in modularity and efficiency since the operator does not need to know the internal mechanisms of the physical network, but can focus, instead, exclusively on the logical operations of the network.

1) Overhead Minimization: In conventional database systems, an elaborate set of concurrency control and recover mechanisms are utilized to provide the ACID property such as two-phase locking, write-ahead-logging, and checkpointing. However, in the network management domain, as discussed in Sections IV and V, weaker forms of the ACID property may be acceptable for certain data categories. In the MANDATE system, we propose to incorporate mechanisms that permit different degrees of concurrency control and recovery to be selectively implemented on a *data partition* basis. By doing so, we expect to realize considerable performance improvements.

Since some categories of structural data (see Section IV-B, V-B) are updated only very rarely and others at a moderate rate, concurrency control is not a major performance issue with respect to transactions having to block while accessing this category of data objects. Yet, it is wasteful to have all transactions pay the computational overhead of invoking the lock manager for each access to a data object given that regulated access is only rarely necessary. Therefore, we plan to use the following solution in MANDATE: For each structural data category, the system maintains a special StructuralCC integer variable which is initially set to 0. Whenever the StructuralCC variable has a value of 0, no concurrency control is employed by transactions accessing that class of structural data. For any other value of StructuralCC, newly arriving transactions have to follow the standard locking protocol. Any transaction that is potentially an updater of the structural data increments the StructuralCC variable at arrival, and decrements the variable when it has finished accessing the structural data. This means that read-only transactions, which arrive when no update transactions are executing, can access their data objects without incurring the overhead of locking. If an update transaction arrives when the StructuralCC variable is 0, the StructuralCC variable is incremented and all the read-only transactions that are currently accessing that category of structural data are aborted and restarted, thus ensuring that they too follow the locking protocol.

In order to enter the updates transmitted by subnetwork managers (see Section V-D) to initiate transactions over the network that directly updates the central MIB. This would require each local manager to know about the implementation of the central MIB and to possess transaction processing mechanisms. Another approach is for the local network managers to merely send the information to the central MIB, which then packages this information into a local transaction that implements the updates in the database. We anticipate that the second approach would be much easier to implement in large networks and, therefore, plan to incorporate it in MANDATE.

2) Data Storage: In Section IV, we described the semantics of the different categories of network management data. A related issue is the choice of storage model, such as the relational model or the object-oriented model, for physically storing the data. Relational models are used in most current database systems since they are compatible with powerful data access languages that are, at the same time, simple and declarative (e.g., SQL) [32]. However, the OSI standards definitions for the MIB interface are based on an object-oriented paradigm [25], which might suggest that an object-oriented model is the appropriate storage model. Note, however, that the OSI standards refer only to the abstract model of management information that is visible at the interface. Therefore, the actual implementation of the persistent storage model could be quite different and is a design choice.

In the MANDATE design, we propose to provide an OSI-compatible object-oriented query interface but leave open, for now, whether a relational model or object-oriented model is used for physical storage. In particular, we note that the use of a relational model for physical storage has merit due to the existence of highly efficient relational data retrieval operators. Algorithms for mapping from an object-oriented interface to the equivalent relational storage model are available in database literature [33]. This approach to MIB design is similar to the common practice among designers of commercial database systems of using the entity-relationship model during the design stage and then converting the final design into a relational model at the physical level [32]. In the NETMATE research project [11], VBase, a commercial object-oriented database system, is used to implement the network model. It is stated in the same paper that one of the future goals of the project is to develop object-oriented network models on top of relational database storage systems.

3) Network Views: The MIB should be able, at all times, to provide the operator(s) with a view of the current state (as best known) of the entire network. This is achieved by combining the current sensor information, the structural information, and the control settings in effect, as shown in Fig. 2. In MANDATE, this idea is generalized to allow the operators to create different views of the network by incorporating a view processor that provides the appropriate view to each operator based on the information in the database. For example, the structural database holds information about the customer subnetworks, which includes details of the physical customer access links and the logical mapping of a customer to the public shared network. An operator trying to find the cause of a customer complaint would use a view, wherein the customer subnetwork is superimposed on the public network to determine whether the fault lies in the public network or is local to the customer subnet. The network views also serve as inputs to the embedded analysis and optimization algorithms.

There is not clear-cut distinction between performance data and fault data, since poor performance can be viewed as a fault. Our definition, however, is that fault data is performance data that is sufficiently critical to appear spontaneously on the operator's console without explicit request; that is, fault data generates an alarm. Therefore, a message about the fault pops up on the operator's console whenever a new fault is either indicated by the sensor data or explicitly sent as an alarm message by a network managed entity.

4) Physical Realization of Logical Objects: Each resource in the managed network, which in OSI parlance is referred to as a *managed object* (MO), will have a corresponding representation in MANDATE. The representations of managed objects will follow established OSI guidelines outlined in SMI/GDMO (Structure of Management Information/Guidelines for the Definition of Managed Objects). However, as outlined in Section IV, MO's are managed through a set of control processes that rely on a three-way partition of MIB data into sensor, structural, and control data. This data partitioning may not always be consistent with OSI MO definitions. For example, it may be desirable to attach to a "structural" object an attribute that is based on sensor information [e.g., the attributes of an object or link might include its end nodes, bandwidth, or propagation delay (structural data) together with the current link utilization (sensor data)]. Where such "cross-overs" occur, we propose that data be physically stored according to the MANDATE classification. This will allow for the most efficient provision of the data-specific database controls described in Section IV. However, the logical views of the data will be consistent with those specified in the OSI MO definitions.

Note that this problem might occur even if the physical partitioning matched that of object definitions, because the network data locations are distributed. For example, as discussed in detail in [19], implementation considerations may dictate that less than a whole managed object be stored in a single system. Therefore, they introduce the notion of *object fragments* and propose techniques for naming and managing of object fragments. Similar issues of "composite managed objects" are discussed in [10].

5) Network–MIB Communications: Information flow (control actions, status information, etc.) between the MANDATE MIB and the network should follow OSI requirements. Thus, MANDATE will offer services according to CMIS and transmit and receive requests through CMIP. However, since most current systems interact through the SNMP protocol, we propose to make our design generic enough such that an SNMP interface could also be integrated into the system.

Of course, the MIB would have to support a naming convention consistent with the communications protocol. Specifically, from an OSI perspective, the heterogeneous network consists of a set of interconnected CME's (conformant management entity). Each CME may be anything from a WAN to a terminal attached to a node, as long as all elements belonging to a particular CME share identical services and protocols. Each CME should have a unique name and address in the MIB, and each network element, i.e., managed object (MO), should be identified by the CME it resides in together with its identifier within that CME.

B. Architectural Model

Although an MIB is logically a centralized repository of all network management data, its physical implementation in large networks will have to be distributed for performance reasons. We assume that there is a central main database which stores all structural data, the most critical sensor data, and the major control settings, while the remaining network state data is stored in the local memories and disks of network components. As shown in Fig. 3, the recently developed concept of a client–server architecture integrates well with this design. In this picture, the DB server is the primary data store site where all updates are synchronized for maintenance of consistency. The network switches periodically propagate status data to the DB server; the switches can also be queried on demand. The client modules are typically workstations for running the application and the user interface software. All database accessing and processing is done at the server. A second server, shown in Fig. 3, is used to mirror the database of the main server and thereby provide fault tolerance. All

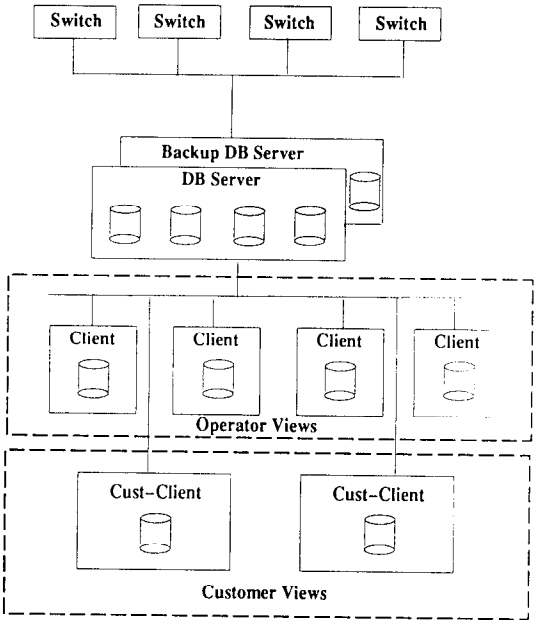


Fig. 3. An enhanced client–server database architecture for the MIB.

updates are made on both servers. During normal operation, query retrieval is load balanced between the two servers. If the primary server happens to fail, the secondary server is promoted to be the primary server for registering updates.

While this client–server architecture *appears* to be an attractive design choice, a straightforward implementation of a client–server architecture creates serious performance problems as it requires enormous processing and I/O on the servers and very large data transmission on the network. For example, if a query is issued at the client, transmitted to and executed at the servers, and if the query's result has to be transmitted in its entirety every time a client needs to access the MIB, a good fraction of the server's and network's capacity would have to be allocated to the support of the database itself rather than the MIB.

To alleviate this problem, we have enhanced the client–server architecture by adding full DBMS functionality on the clients for caching and processing views (subsets) of the MIB data [28]. At any point in time, a particular view is maintained in each client. In many cases, the client workstation will be supporting an operator who is responsible for real-time control decisions. Therefore, updates to the MIB must be propagated to the client views in real time. The client and server DBMS's cooperate and split the task of query processing. By placing client workstations at strategic nodes on the network, database access is performed in parallel from multiple client databases, thereby alleviating the bottleneck at the primary server. Some of these client workstations may be located at customer sites, as shown in Fig. 3.

Having data downloaded on a large number of clients can increase performance, provided that the overhead of maintaining consistency amongst them is manageable. First, the activities of synchronizing and refreshing downloaded data are repeated very often in a network; therefore, as the number of sites increases, the processing throughput can rapidly deteriorate due to blocking. Second, because sites collectively carry a large variety of data subsets, each subset is pertinent to a client's function, place of deployment, and response requirements. The database servers waste most of their capacity in keeping track of who-needs-what-when in order to propagate changes that affect individual clients. On the other hand, if all changes are broadcast to all clients, the clients would be incapacitated by having to check all relevant and irrelevant updates (most of them are irrelevant to each individual client). Therefore, the intelligence–namely who-needs-what-when–must be distributed to the client workstations, which would selectively request only relevant updates from the logs maintained at the servers. In the following subsections, we show how the problems can be resolved by using incremental update algorithms.

1) Incremental Computation Models: We present the recently developed concept of incremental computation models [26], [27]. Conventional computation models are based on reexecution; that is, the entire computation is repeated each and every time results are needed. Nothing is retained from previous executions and, often, even the optimization of the computation is repeated. In contrast, incremental computation models utilize cached results or access paths to generate these results [26]. With these models, the results of subsequent accesses to the same portion of the database are realized by applying the computation to the input differentials, rather than reexecuting the computation on the whole input.

The following example illustrates this concept. Consider the following two tables:

- C=Cust_call(cust_id, call_id, route_id)
- R=Route(route_id, link_id)

where C stores information about customer calls and R stores the routes that these calls go through. Assume that a customer representative needs to know which customers use what links so that, if a link goes down, the affected customers can be identified. This requires computation of the join C*R. In the incremental model, the C*R join is computed only once, and thereafter maintained incrementally by using only the differential files δC and $\delta R, \delta(C * R) = (\delta C * R) \cup (C * \delta R) \cup (\delta C * \delta R)$ (here, C and R refer to the updated versions of the relations). The records of query results are not stored explicitly but, instead, use a storage structure called ViewCache [26]. It consists of a collection of pointers to the underlying base table records and can quickly be dereferenced to generate the actual records. ViewCache is both compact and efficient for the incremental update of the pointers.

Incremental computation can be performed on demand, periodically, or immediately. Each invocation is performed on small input increments, which permits a significant reduction of the response time. The cost of computing is amortized over the lifecycle of the computed information, a concept that is absolutely orthogonal to the reexecution model of transient and nonpersistent software. Since computation is done in increments, performance is improved by several orders or magnitude.

2) Incremental Client–Server Architecture: The previously mentioned problems associated with distributed network data-

base processing in client–server architectures can be resolved by integrating the enhanced client–server architecture with the incremental computation model. The database activities, which include I/O and processing, are now distributed between the server and the requesting client. A portion of I/O and processing is related to the incremental updates of the cached data done at the server, leaving all the rest to be done on the client. Both I/O and processing on multiple clients can be done in parallel, and is the major factor in the significant increase of overall throughput. A recent study has shown that the client–server architecture, in association with the incremental computation model, achieves two orders of magnitude performance increase over a standard client–server database architecture [9]. More importantly, the incremental model seems especially appropriate for real-time maintenance of network views, which are continuously being used by the network operators and customers. This is because, with this model, keeping the views up-to-date requires only incremental computations and relieves the system from the burden of having to recompute the entire view in each refresh cycle. Since, under normal network operations, the views change only very slowly over time, the incremental model can realize great improvements in performance and minimize the overhead of maintaining the views on the other system functions.

In MANDATE, we intend to use the incremental client–server database architecture to provide the following: 1) Real-time refreshing of network views; 2) Immediate update propagation from the primary site to its secondary backups—this guarantees that, in case of failure, the secondary site is up-to-date for promotion to primary server; 3) A quick switch capability between a failing primary site and its backup secondary one; 4) Parallel access of dynamically distributed data on the enhanced clients and significant reduction on the server's I/O; 5) Preservation of the appropriate level of centralized control.

In summary, the enhanced incremental client–server architecture provides the functional advantages of a distributed architecture while retaining the high performance required to support the MIB.

C. Network Control

The fundamental goal of network management is to be able to control the state of the network. In the context of an MIB, a network control process is any mechanism that makes the network respond to stimuli collected by various network sensors. Examples of such stimuli include traffic data along links (e.g., load factors, retransmission rates), switching information (e.g., queue lengths, throughput), and network faults. In current large networks, numerous sensors continuously monitor all aspects of the network operation and generate vast quantities of data. Though most of this data is routine, events such as link failures and switch malfunctions may occur which require remedial action by the network management system.

For network MIB systems, we classify network control mechanisms along two dimensions: local versus global and automatic versus manual. We present brief descriptions of each class with illustrative examples.

1) Local Control: Local control mechanisms rely on local data collection and local decision models. By local, we refer to specifc components of the network as opposed to the network as a whole. An example of a local network control mechanism is the classical window-based flow control scheme for regulating traffic between a source destination pair [12]. In this protocol, incoming packets are accepted as long as the number of unacknowledged packets is below a prespecified threshold. If the threshold is exceeded, a local control mechanism turns off the acceptance of further traffic until a sufficient number of outstanding packets are acknowledged. The advantage of local controls is that they incur little or no communication overhead, since decisions are made locally with local data, and minimal information exchange is involved. Due to this locality of operation, local control processes are unaffected by remote network failures and network congestion.

The logic for a local control process would reside completely within the local network element, e.g., switch, line control unit, etc. However, this process would be a logical part of the MIB, and its parameters could be modified via MIB update. Specifically, the MIB would contain variables corresponding to the local controller parameters. An update to one or more of these database variables would trigger a message from the MIB to the local element, specifying that the corresponding change be made in the parameter value.

2) Global Control: Global control processes rely on network-wide data and/or global decision models. Examples of global control mechanisms include routing algorithms that compute routes based on network-wide traffic estimates. Clearly, global control processes are capable of optimizing network-wide performance characteristics. However, they are more vulnerable to network failures and have greater information overhead since decisions must be communicated across the network.

A global control action could be specified by an operator or could be the result of an automatic process. The operator or process would implement the control action by updating certain database variables. These updates would trigger messages (OSI M-ACTION requests) to the local network elements specifying that the appropriate actions be taken.

It should be noted that a significant amount of research has been devoted to the development of distributed implementation of network algorithms [15], [31]. These implementations attempt to develop local controls that approach global performance standards.

3) Automatic Control: Automatic controls monitor certain network performance data. When specific conditions are met, control settings are automatically changed without operator intervention. For example, assume that a particular switch has a local control process that equally allocates incoming calls between a source destination pair among three different routes. If one of these routes becomes overloaded, an automatic control process embedded in the switch is triggered such that only 20% of all new traffic is sent along the saturated route and the remainder is split equally among the two other routes. Automatic controls can be either global or local.

Automatic local controls are in abundance today and would remain relatively unchanged in a MANDATE environment. Automatical global controls would be based on two sets of active database elements. First, triggers would sense a network state indicating that the control process should be initiated and would invoke the appropriate algorithm. Second, as explained, the result of the global control algorithm would be to update certain database variables. A second set of triggers would dispatch the messages to the appropriate network elements.

4) Manual Control: Manual control processes either permit or require human intervention. Network operators alter control settings in the network using these processes. In a sense, manual controls represent one of the major justifications of the MIB. Clearly, the role of the MIB is to provide the network manager with information that supports decision making regarding the setting of control parameters. This supporting activity may be achieved passively by simply providing an interface between the network operator and network status information. Alternatively, it may be achieved through an alarm system that notifies the network manager of network conditions that require actions on his or her part. As in the previously described cases, operator-mandated controls would be carried out by an operator update to the appropriate database variable.

D. Embedded Optimization and Analysis Algorithm

One of the most significant potential advantages of the MANDATE design is the ability to carry out sophisticated global analyses based on network-wide data. To achieve this objective, the design should provide a set of flexible mechanisms for interfacing optimization and analysis algorithms with the MIB. In this context, we use the term *optimization algorithm* to refer to a process that outputs a set of values for certain control variables and the term *analysis algorithm* to refer to a process that evaluates the impact on network performance of a set of control settings input into the algorithm. Thus, most types of simulations would be classified as analysis algorithms. However, faster and simpler procedures, which may be more appropriate in the context of real-time decision making, are also included. The embedded optimization and analysis features are expected to help network operators make higher quality control decisions. Similar facilities are provided in the NETMATE system [11].

We now discuss the interaction between embedded optimization and analysis algorithms and the MIB. It is important to note that the network control processes, which were discussed in the previous subsection, are different from embedded algorithms. Network control processes comprise the entire mechanism of controlling the process of network management. As such, they include the process of data collection by network sensors, the invocation of daemons under certain conditions detected by the sensors, the execution of control algorithms that accept the data as input and perform computations to derive control settings, the potential screening of the settings by human users, the potential execution of analysis algorithms that output projected network performance based on the control settings, and the selection of the final settings and final execution of these settings. Thus, network control processes are complex mechanisms of which optimization and analysis algorithms are only components. However, optimization algorithms are an integral component of the global controller.

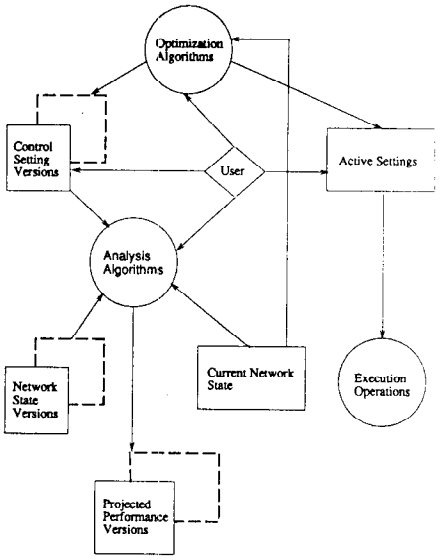


Fig. 4. Interactions between algorithms and the MIB.

Fig. 4 gives a more detailed view of the interaction between the algorithms and the MIB. Examples of optimization algorithms include algorithms that output routes, flow controls, and access controls. In general, the optimization algorithm draws input from the MIB and outputs control settings. The inputs to the algorithm consist of the current network state, as embodied by some network view, and the existing control settings. We also envision the presence of achieved control settings in the MIB, which consist of old control settings and the corresponding network states for which they were used. These also may serve as input to optimizing the routines which may find commonality between the current network state and some old state, which could indicate that the corresponding control setting is appropriate under the current conditions.

The output from an optimization algorithm will be used to update either an active control variable or a nonactive control variable. An update of an active control variable would trigger the immediate execution of a control action as was described in the previous section. Nonactive control variables would usually be examined by users prior to making a final decision on a control action. That is, if the user wished to use the control setting, he or she would set the value of the active variable to the value of the corresponding nonactive variable. Whether the output of an algorithm is direct to an active or nonactive variable is determined by the trigger that invoked the execution of the algorithm. For example, a unstable network state may trigger the algorithm to automatically take action by updating an active variable, while a less critical situation may induce the update of a nonactive variable and the subsequent examination of the proposed control setting by the operator. In the latter case, the operator may wish to run the algorithm several times, possibly with different parameters, and compare the output. Typically, in that case, as a result of multiple invocations of the algorithm, there would be several versions of control settings

present leading to the need of a version control scheme.

The role of an analysis algorithm is to evaluate the performance effects of a proposed control setting. The inputs to the analysis consist of a network state, an existing control setting, and the proposed control setting. The output is a detailed analysis of the effect on performance of using the proposed control setting. A typical analysis algorithm might be a queueing model which evaluated the effect of using a particular route structure (the control setting) with a particular traffic pattern (the network state). Such an analysis might make use of simulation or, alternatively, other models might be appropriate particularly if very fast response were required.

The general environment that we propose to support in the MANDATE system is one in which a user can invoke a given optimization algorithm several times to produce alternate solution versions (proposed control settings). The user would typically examine and possibly modify various versions and may also wish to invoke an analysis algorithm that produced a detailed evaluation of one or more of the proposed control settings. Finally, the user would choose one of the versions as a "final version" which, in MANDATE, would correspond to making the control setting active.

E. Temporal Requirements

An important functionality that has to be provided by a network MIB is that of supplying temporal views of the state of the database; that is, a description of the network state as of a specific point in time. This is necessary in order to conduct postmortem fault analysis or to profile performance trends. Therefore, the network MIB must support the retrieval and analysis of network data describing the state of the network as was known at a particular location and time. In addition, the system must support the analysis of change of network state data in order to provide information describing how the network behavior at a particular location changed over time. Such information constitutes an important basis for long-term planning and fault analysis.

In typical conventional database systems, transactions are usually processed in a first-come first-served manner and the objective is to minimize average transaction response times. In a network management database system, however, data from numerous sources electronically arrives into the system in a continuous fashion [1]. Due to the high rate of data arrivals, delays in processing the data streams will cause the data streams to back up and the database will cease to provide an accurate picture of the network state. Therefore, in contrast to a conventional DBMS where the goal usually is to minimize transaction response times, the emphasis in the network MIB is on processing the database updates in real time.

We discuss how MANDATE proposes to provide temporal view to the network operators, and how the network state information is installed in the database in real time.

1) Temporal Views: A requirement for transactions that access network state information is that they have to enforce temporal consistency; that is, the data items they read should have existed in the real world at approximately the same time. For example, using the link utilizations that were valid ten seconds back in association with the node queue lengths that were measured ten minutes ago clearly makes no sense. Therefore, the "right" versions of data items have to be used in processing each query. Note that temporal consistency is evaluated with respect to the real-world time that the data was valid, not the time at which the data was installed in the database. (These two time concepts are referred to as *valid time* and *transaction time,* respectively, in the database literature [29].) As a consequence, the source and time of measurement of network data are important attributes of data collected in a network management system.

Two types of temporal queries are possible: a) *Historical queries:* These queries expect their answer to be based on the real-world network state as of some time in the past. b) *Current queries:* These queries expect their answer to be based on as recent a real-world network state as possible. In order to help maintain temporal consistency for queries, each new version of a data item is timestamped with the real-world time of occurrence of the event or measurement. The valid time of the kth version of data item d is denoted by $E_k(d)$. Also, the system maintains a table describing the periodicity P_d of updates for each data item d that is periodically updated by the network sensors. When a user submits a historical query that requires the network state as of some earlier time T_p, for each data item, the version V_k of the data item which satisfies the constraints $E_k(d) \leq T_p$ and $E_{k+1}(d) > T_p$ is used for computing the answer. For periodically updated data, if the $k+1$th version has not yet been installed in the database, the V_k version is used if $T_p \leq E_k(d) + P_d$. Otherwise, the query processor waits for the V_{k+1} version to be installed and then uses it. If blocking cannot be tolerated, however, then an alternative is to rerun the query as of an earlier time than T_p, informing the operator of this change. A guaranteed earlier time for which the query will run through without blocking for unavailable data is $t - \text{Max}(P_i)$ where t is the current time and P_i are with reference to the periodically updated data items that the query wishes to read. This scheme can be used for processing current queries in a timely manner.

2) Real-Time Response: There are two types of network updates that arrive to the MIB: *periodic* and *sporadic*. The periodic data is the performance data that arrives in a regular fashion, whereas the sporadic data is the billing and fault information which arrives in random fashion based on the network's use and operational status. Since the sporadic data is critical data that has to be registered in the database, higher priority will be given in MANDATE to processing these updates as compared to processing the periodic updates. In addition, a dedicated processor with sufficient excess capacity will be used in MANDATE in conjunction with Earliest-Deadline scheduling to ensure that when the sporadic updates are arriving at their normal rate, they will be registered in the database before the expiration of their deadlines.

Under normal operations, when the sporadic updates are arriving at their usual rate, MANDATE will also ensure that all the periodic updates are entered into the database system. This is done in the following manner: For each periodic sensor S_i, the system maintains information about its computation requirement C_i and period P_i. The deadline for each of these

periodic update transactions is the time of the next triggering; that is, the taks must be completed prior to the instant of its next occurrence. In order to ensure that all the periodic updates are registered, it is sufficient to use a coprocessor with sufficient speed such that $\Sigma_{i=1}^{N} C_i/P_i \leq 1$ in conjunction with earliest deadline scheduling [20].

Under periods of stress loading or emergency situations, which is when the sporadic updates arrive at a much higher rate than normal, the system starts to miss the deadlines of periodic updates since sporadic updates have higher priority than periodic updates. To minimize the number of missed deadlines under these overload conditions, we propose to use the Adaptive Earliest Deadline scheduling policy described in [16], which has been shown to provide robust good performance under a variety of workloads and system loading conditions.

VII. Conclusions

The Management Information Base (MIB) of a network management system is a critical component since it provides the interface between all functions of the network management system. In this paper, we introduced the design of MANDATE, a proposed database system for effectively supporting the management of large networks. The MANDATE design aims to provide network operators and customers with an MIB interface that allows them to control and evaluate the network operations by interacting solely with the database. The actual implementation of the control decisions in the physical network are expected to be handled by MANDATE's internal processes.

We showed that the network management environment is a specialized application area with unique characteristics that can best be taken advantage of by an MIB that is designed specifically for this environment. To this end, MANDATE's design was built bottom-up, unlike other designs discussed in the literature which have, for the most part, added network-related modifications on top of commercially available general-purpose database systems.

The proposed underlying structural framework of MANDATE is a client–server architecture, which is enhanced by the use of a DBMS functionality on the clients, and an incremental computational model. This architecture is expected to deliver both high-performance and functional advantages of a distributed architecture.

We presented a detailed analysis of the different categories of data that are stored in the MIB. Based on this analysis, we showed that concurrency control and recovery protocols, which are fundamental mechanisms in conventional database systems, are not necessary for all categories of network management data. This insight is used in the MANDATE design to selectively eliminate concurrency control and recovery overheads. We expect that these optimizations will result in significantly improved real-time performance.

We propose to provide a rich variety of control structures in the MANDATE system. Both local control mechanisms, which control specific network components, and global control mechanisms, which control network-wide performance, will be supported. For each control mechanism, the user can either control it manually or have the system control it automatically. We also plan to provide MANDATE support for embedded network analysis and optimization algorithms, which are essential for deriving high-quality control decisions.

Network operators and customers will interact with MANDATE through a view-based interface. Both customers and operators obtain views of the network that are constructed according to their individual requirements by the view processor of the MANDATE system. These views will be maintained on a real-time basis, an essential feature for viable network management systems. In addition, MANDATE will provide the network operators with a facility for obtaining historical views of the database state. These views are extremely useful for postmortem analysis of the causes for network faults and to derive long-term performance profiles.

In summary, the MANDATE design proposes to use special characteristics of network management data and transactions, together with recent advances in database technology, to efficiently derive its functionality. Currently, MANDATE is still a paper design in which we have attempted to identify the basic mechanisms that need to be incorporated in network MIB's. As part of our future research, we plan to test and tune the MANDATE design by implementing it on a network testbed and to follow this up with a detailed performance study.

References

[1] L. Wasson, B. Schwab, and J. Sholberg, "Database management for an integrated network management system," in *Network Management and Control*, A. Kershenbaum and M. Malek, Eds. New York: Plenum, 1990.

[2] L. Ball, *Cost-Effective Network Management*. New York: McGraw-Hill, 1992.

[3] M. O. Ball, A. Datta, and R. Dahl, "Building decision support systems that use operations research models as database applications," Tech. Rep. SRC-TR 92-109, Inst. for Syst. Res., Univ. of Maryland at College Park, 1992.

[4] S. Bapat, "OSI management base implementation," in*Integrated Network Management II*, I. Krishnan and W. Zimmer, Eds. North Holland, 1991, pp. 817–831.

[5] P. Bernstein, V. Hadzilakos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Reading, MA: Addison-Wesley, 1987.

[6] Sprint Network Management Center, Site Visit, Apr. 1992.

[7] M. Chernick, K. Mill, R. Aronoff, and J. Strauch, "A survey of OSI network management standards activities," Tech. Rep. NMSIG87/16 ICST-SNA-87-01, Nat. Bureau of Stand., 1987.

[8] Digital Equipment Corporation, "Enterprise management architecture—General description," Tech. Rep. EK-DEMAR-GD-001, 1989.

[9] A. Delis and N. Roussopoulos, "Performance and scalability of client–server database architectures," in *Proc. 18th Int. Conf. Very Large Data Bases*, Aug. 1992.

[10] A. Dittrich, "Composite managed objects," in *Integrated Network Management II*, I. Krishnan and W. Zimmer, Eds. North Holland, 1991, pp. 789–799.

[11] A. Dupuy, S. Sengupta, O. Wolfson, and Y. Yemini, "NETMATE: A network management environment," *IEEE Netw.*, Mar. 1991.

[12] A. Ephremides and S. Verdu, "Control and optimization methods in communication network problems," *IEEE Trans. Aut. Contr.*, vol. 34, no. 9, 1989.

[13] E. Ericson, L. Ericson, and D. Minoli, "Expert System Applications in Integrated Network Management." New York: Artech, 1989.

[14] K. Eswaran *et al.*, "The notions of consistency and predicate locks in a database system," *Commun. ACM*, vol. 19, no. 11, 1976.

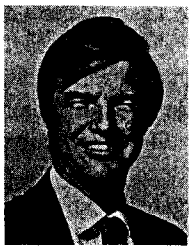
[15] R. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Trans. Comput.*, vol. 23, pp. 73–85, 1977.

[16] J. Haritsa, M. Livny, and M. Carey, "Earliest deadline scheduling for real-time database systems," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 1991.
[17] G. Held, *Network Management (Techniques, Tools and Systems).* New York: Wiley, 1992.
[18] S. Klerer, "The OSI management architecture: An overview," *IEEE Netw.*, vol. 2, no. 2, 1988.
[19] S. Klerer and R. Cohen, "Distribution of managed object fragments and managed object replication: The data distribution view of management information," in *Integrated Network Management II,* I. Krishnan and W. Zimmer, Eds. North Holland, 1991, pp. 763–773.
[20] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM,* Jan. 1973.
[21] M. Stonebraker, *The Ingres Papers.* Reading, MA: Addison Wesley, 1986.
[22] Oracle Corporation, *Oracle Users Guide,* 1983.
[23] R. Elmasri and S. Navathe, *Fundamentals of Database Systems.* Redwood City, CA: Benjamin Cummins, 1989.
[24] C. Rauh, "The concept of the network management information base in CNM, and TRANSDATA network management scheme," in *Integrated Network Management II,* I. Krishnan and W. Zimmer, Eds. North Holland, 1991, pp. 833–844.
[25] M. Rose, *The Simple Book—An Introduction to Management of TCP/IP Based Internets.* Englewood Cliffs, NJ: Prentice Hall, 1991.
[26] N. Roussopoulos, "The incremental access method of View Cache: Concept and cost analysis," *ACM Trans. Database Syst.,* vol. 16, no. 3, 1991.
[27] N. Roussopoulos, N. Economou, and A. Stamenas, "ADMS: A testbed for incremental access methods," *IEEE Trans. Knowledge and Data Eng.,* 1993.
[28] N. Roussopoulos and H. Kang, "Principles and techniques in the design of ADMS±," *IEEE Trans. Comput.,* vol. 19, 1986.
[29] R. Snodgrass and I. Ahn, "A taxonomy of time in databases," in *Proc. ACM SIGMOD,* June 1985.
[30] K. Terplan, *Communication Networks Management.* Englewood Cliffs, NJ: Prentice Hall, 1992.
[31] J. Tsitsiklis and D. Bertsekas, "Distributed asynchronous optimal routing data networks," *IEEE Trans. Automatic Contr.,* vol. 31, pp. 325–332, 1986.
[32] J. D. Ullman, *Principles of Database and Knowledge-Base Systems.* New York: Computer Science Press, 1988.
[33] P. Valduriez, S. Khoshafian, and G. Copeland, "Implementation techniques of complex objects," in *Proc. 3rd Int. Conf. Very Large Data Bases,* Aug. 1977.
[34] R. Valta, "Design concepts for a global network management database," in *Integrated Network Management II,* I. Krishnan and W. Zimmer, Eds. North Holland, 1991, pp. 777–788.
[35] O. Wolfson, S. Sengupta, and Y. Yemini, "Modeling network management functions as database operations," Tech. Rep. CUCS-038-90, Dept. Comput. Sci., Columbia Univ., 1990.

Jayant R. Haritsa received the B.S. degree in electronics and communications engineering from the Indian Institute of Technology, Madras, in 1985, and the M.S. and Ph.D. degrees in computer science from the University of Wisconsin, Madison, in 1987 and 1991, respectively.

During 1991–1992, he was a Post Doctoral Fellow at the Institute for Systems Research, University of Maryland, College Park. He is currently an Assistant Professor in the Supercomputer Education and Research Centre at the Indian Institute of Science, Bangalore, India. His research interests include database systems, real-time systems, network management, and performance modeling.

Dr. Haritsa is a member of ACM.



Michael O. Ball received B.E.S. and M.S.E. degrees in engineering science from Johns Hopkins University, and the Ph.D. degree in operations research from Cornell University.

He holds a joint appointment in the Institute for Systems Research and the College of Business and Management at the University of Maryland, College Park. His research interests are in the areas of network optimization and network reliability analysis, particularly applied to the design of telecommunications networks and transportation systems. He has published extensively on these topics in a variety of journals. Hs is an Area Editor for *Operations Research,* and is Associate Editor for *Networks, Operations Research Letters* and *IEEE Transactions on Reliability.* He is currently co-editing a volume on the *Handbook of Operations Research and Management Science.* Prior to coming to the University of Maryland, he was a member of the technical staff at Bell Laboratories. He has also visited, for extended periods of time, the Center for Operations Research and Econometrics (CORE) in Louvain, Belgium, the Institute for Economics and Operations Research in Bonn, West Germany, and the Department of Combinatorics and Optimization in Waterloo, Ontario. During the 1992–1993 academic year, he visited the Department of Operations Research at the University of North Carolina, Chapel Hill.

Nicholas Roussopoulos received the B.S. degree from the University of Athens, Athens, Greece, and the M.S. and Ph.D. degrees from the University of Toronto, Toronto, Canada.

He is a Professor in the Department of Computer Science and the Institute of Advanced Computer Studies at the University of Maryland. He served on the Space Science Board Committee on Data Management and Computation (CODMAC) from 1985 to 1988. He was the General Chairman of the ACM International Conference on Data Management (1986). He has also organized and chaired a series of workshops for the VHSIC Engineering Information System program. He also serves on the Editorial Boards of three international journals, information systems, decision support systems, and intelligent and cooperative information systems. He has published over 50 refereed papers. His research area are in database systems, multiple databases and interoperability, engineering information systems, geographic information systems, expert database systems, and software engineering. He has worked as a Research Scientist at IBM, San Jose, CA, and with the Department of Computer Science, University of Texas, Austin.

Anindya Datta received the B.S. degree from the Indian Institute of Technology, Kharagpur, India, in 1986 and the M.S. degree from the University of Maryland.

He is a Ph.D. candidate in information systems and operations research at the University of Maryland, College Park, and is part of the interdisciplinary research team at the Institute of Systems Research. His research interests include telecommunication network management, databases, and algorithms.

Mr. Datta is a member of ACM and ORSA.

John S. Baras, photograph and biography not available at the time of publication.