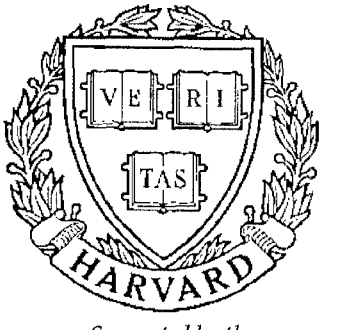
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

MANDATE: MAnaging Networks using DAtabase TEchnology

by J.R. Haritsa, M.O. Ball, N. Roussopoulos, and A. Datta

TR 92-98

MANDATE: MAnaging Networks using DAtabase TEchnology

Jayant R. Haritsa Michael O. Ball
Nicholas Roussopoulos Anindya Datta

Systems Research Center
University of Maryland
College Park, MD - 20742

ABSTRACT

In a recent opinion poll of telecommunications executives, enterprise network management was identified to be the top technological issue of the future. At present, however, there do not exist any viable solutions to this critical problem. Therefore, considerable research efforts are being focused on the development of effective network management tools. 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 describe MANDATE (MAnaging Networks using DAtabase TEchnology), a database system that is designed to effectively support the management of large networks of the future. MANDATE uses special characteristics of network management data and transactions, together with recent advances in database technology, to efficiently derive its functionality.

1. 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 sub-networks such as LANs (Local-Area Networks), MANs (Metropolitan-Area Networks), WANs (Wide-Area Networks), etc. 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 sub-networks 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 sub-networks may have different vendor-specific network management systems. Third, the information being transmitted over the network may be "multi-media", that is, semantic differences exist in the types of the 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 the above-mentioned reasons, future enterprise networks are expected to be highly complex in their transmission, performance, and communication characteristics. Due to this complexity, and due to the disparity of management systems for individual subnetworks, efficient management of an enterprise network is an extremely challenging problem. At present, however, there do not exist any viable solutions to this critical problem, which has been identified, in a recent poll of telecommunication managers [Bort88], as the most important technological issue of the current decade. Therefore, there is a clear need for research and development of network management tools. In fact, [Rose91] claims that lack of a workable network management system is the key reason for preventing the deployment of large OSI networks.

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. [Valt91, Bapa91, Terp92, Ball92]). The database is required to store

information on network and system configuration, current and historic performance, trouble logs, security codes, accounting information, etc. [Kler88]. In OSI parlance, this database is called a Management Information Base (MIB). A practical example of a MIB-based architecture is DEC's EMA (Enterprise Management Architecture), where a Management Information Repository is defined as a central component of the Director [DEC89]. 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 a MIB. In this paper, we describe the design of MANDATE, a MIB system for effectively supporting the management of large enterprise networks. Currently, MANDATE is a paper design; however, we shortly plan to test and tune the design by implementing it on a "toy" network and then follow up with a detailed performance study.

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[1]. 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 system is that it is a complete 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. In addition, this research approach, by identifying those features of database system design that are critical to network management, helps industry network management groups in choosing the most appropriate currently available commercial database system for their short-term needs. Finally, MANDATE uses special characteristics of network management data and transactions, together with recent advances in database technology, to efficiently derive its functionality.

The remainder of this paper is organized in the following fashion: In Section 2, the role of database systems in network management is discussed in detail. In Section 3, the related work on database support for network management is briefly reviewed. Then, in Sections 4 and 5, we describe the design of the MANDATE database system. Section 4 focuses on the data and transaction modeling aspects of MANDATE, while Section 5 details how MANDATE provides the required functionalities. Finally, in Section 6, we summarize the main conclusions of the study and outline future research avenues.

---

[1] This approach is also known as the "data-centric model of network management".

2. ROLE OF DATABASES IN NETWORK MANAGEMENT

The ISO/ANSI standards committee [Cher87] has classified the sophisticated functionality required of network management systems into six categories, as described in [Feri88]:

(1) Configuration Management: Defining, monitoring, and controlling network resources and data;

(2) Fault Management: Detecting, diagnosing, and recovering from network faults;

(3) Performance Management: Tracking and analyzing current and long-term network performance;

(4) Security Management: Ensuring only secure and authorized access to network resources;

(5) Accounting Management: Recording usage of network resources and generating billing information;

(6) Directory Management: Supporting directories for managing network assets and user information;

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. Although this functional classification has been introduced primarily for the future OSI-based network environment, it has received acceptance from the current vendors of network management products [Feld89].

There is considerable overlap and interaction between the various management sub-systems described above. 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 co-ordinate their activities, a common "public workspace" or *database* is necessary. Therefore, *a logically integrated database is the heart of a network management system* [Schw90] – it provides the interface between all functions of the network management system, as shown in Figure 2.1. 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.

2.1. Requirements on MIB

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

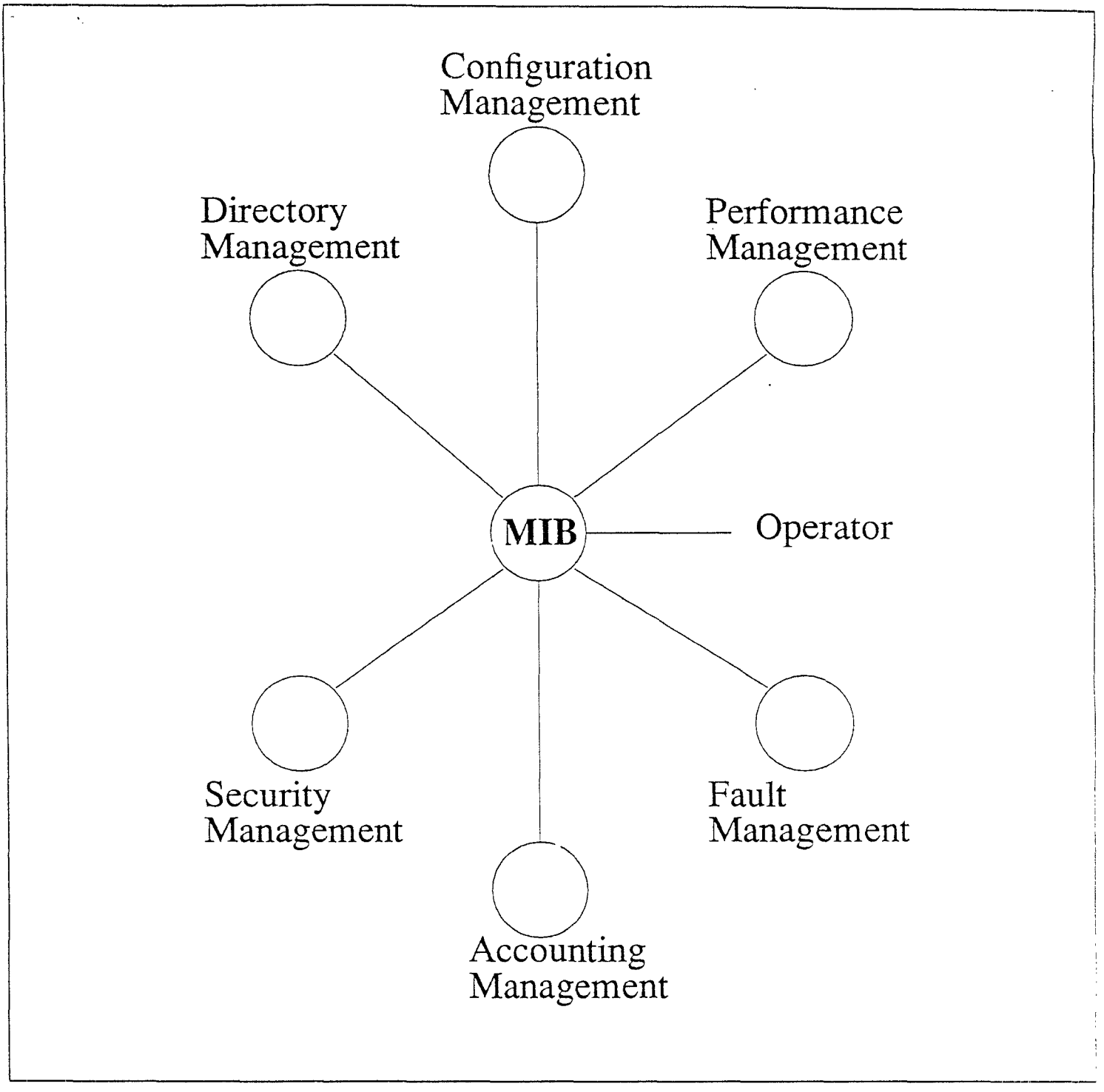


Figure 2.1: Network Management Model

(1) Homogeneous interface: Present a uniform interface to the operator that is independent of the individual sub-network 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 sub-networks or increase the functionality of existing sub-networks 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 simulations), 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 the above list, we see that a 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.

2.2. 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, INGRES) should be sufficient for implementing a network management MIB. There are several reasons, however, due to which existing DBMS products are not

satisfactory from the network management perspective:

(1) Standard off-the-shelf DBMSs lack many of the required MIB functionalities, such as the real-time capabilities and decision support facilities described above.

(2) Conventional DBMSs have been developed for the commercial query processing environment and are primarily geared towards monetary 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.

(3) The objective of conventional DBMSs is to efficiently implement a transaction model that provides the properties of atomicity, serializability, and permanence. 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 implemented for the database to be correct, and this guarantee is provided by conventional DBMSs. 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. A MIB can use this property of network data to derive some of its functionality. For example, under over-load 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. In this paper, we describe the design of MANDATE, a MIB that is *tuned* to the task of enterprise network management.

3. 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.

In [Schw90], issues similar to those addressed in this paper were considered. The focus in that work was on evaluating how conventional relational DBMS packages would serve in the role of a 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.

An overview of the issues involved in implementing the MIB interface definition laid down by the OSI standards committee was presented in [Bapa91]. 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 a MIB, it does not, however, provide a detailed design for achieving these functionalities.

A Layered Attributed Graph was proposed as a formal mechanism to model a network in [Valt91]. 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. [Terp92, Held92, Ball92]). 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 the mechanisms for realizing the functionality requirements.

Finally, there are numerous papers on expert systems for network management (see [Eric89] 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.

4. DESIGN OF MANDATE

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 actions that update the corresponding data objects in the MIB. The *actual* implementation of these changes in the physical network are made by execution processes that are activated or triggered by the database system. This 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.

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. In this section, we discuss the data and transaction modeling aspects of the network environment.

4.1. Data Model

Network management data can be broadly classified into three types: Sensor Data, Structural Data, and Control Data, as shown in Figure 4.1, which describes a high-level abstraction of the MIB

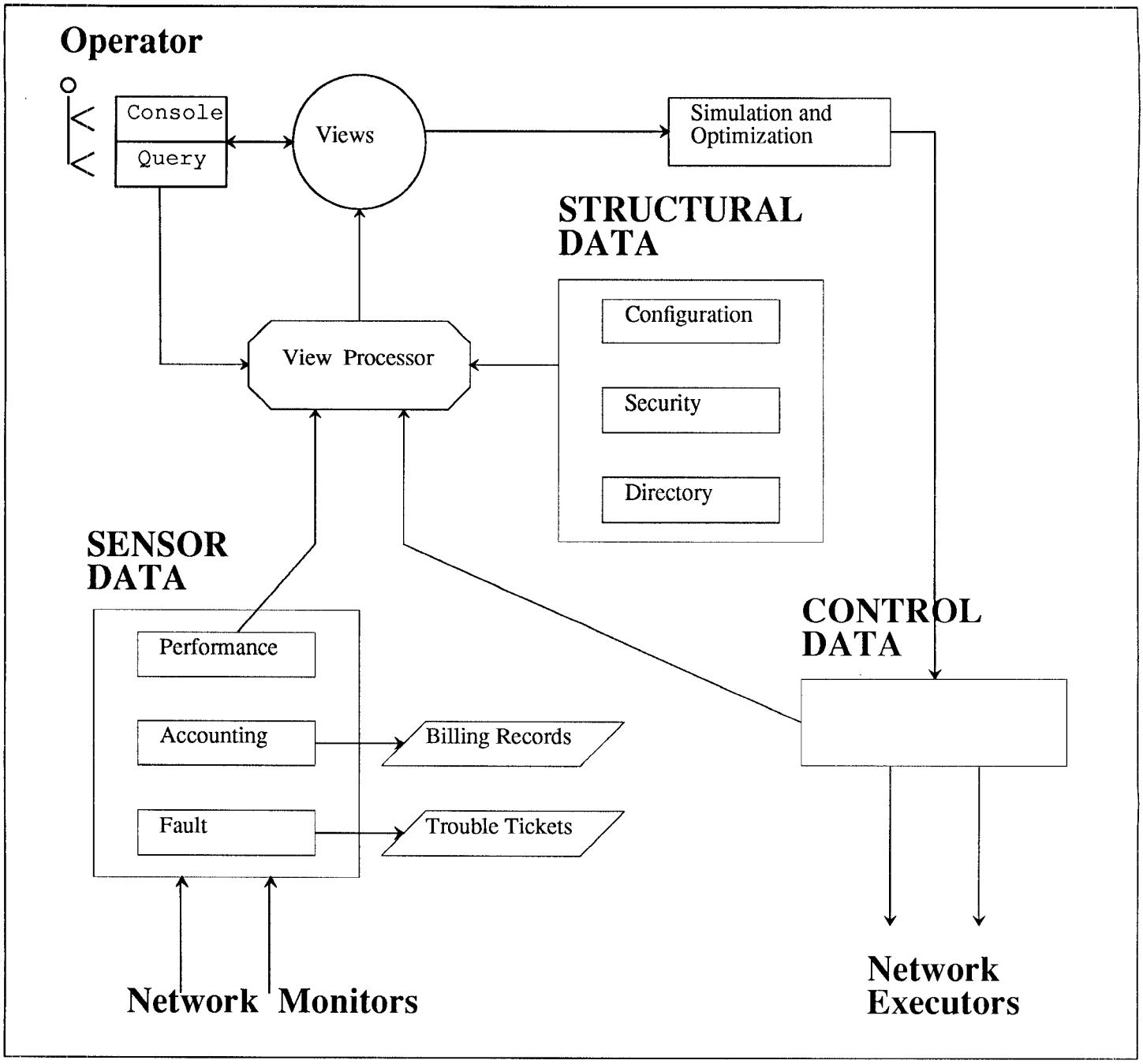


Figure 4.1: Model of MIB

data model. As explained in detail below, 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.

4.1.1. Sensor Data

The sensor (or measurement) data is the raw information that comes in from the network monitoring processes, and includes variables such as node queue lengths, retransmission rates, link status, call statistics, etc. 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 [Rose91]. 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) similar to those provided by commercial DBMSs.

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 while 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 "sacred" – here individual samples may not be essential, and 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 might be useful to retain the *history* of values for long-term post-mortem 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 percent). 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, the number of retransmissions, etc. and not version 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 to 30 gigabytes per day [Sprin92].

4.1.2. 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, the customer description records, etc. 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 typically changed only in response to significant network events such as adding a new switch to the system, or offering a new type of customer service, 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, etc. are represented, and may be of the order of several gigabytes.

4.1.3. Control Data

The final data category is the 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, the routing table, etc. 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 pre-defined 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 traffic 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 data and the control data. However, since these changes are rather infrequent (especially for the structural data), we expect that the overhead of maintaining the update history will not be significant.

4.1.4. Data Storage Model

In the above discussion, we have 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) [Ull88]. However, the OSI standards definitions for the MIB interface are based on an object-oriented paradigm [Rose91], 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.

The OSI definition of the MIB object class hierarchy, when graphed as an inheritance tree, is broad and shallow [Bapa91]. Therefore, the inheritance mechanism of the object-oriented model, which is its primary virtue, does not really come into play. Consequently, in the MANDATE design, we provide an OSI-compatible object-oriented query interface but use a *relational* model for the physical storage in order to gain processing efficiency. Algorithms for mapping from an object-oriented interface to the equivalent relational storage model are available in the database literature [Vald86]. 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 [Ull88].

4.2. Transaction Model

Having discussed the characteristics of network management data, we now move on to considering the various types of transactions that operate on the sensor, structural, and control databases. The traditional view of a transaction in commercial database systems is that it provides the so-called *ACID*

property, that is, atomicity, consistency, isolation, and durability. In order to provide the ACID property, an elaborate set of mechanisms such as write-ahead-logging, locking, checkpointing, etc. are utilized in these database systems. However, in the network management domain, weaker forms of the ACID property may be acceptable for certain data categories. By identifying and taking advantage of these less restrictive forms, performance improvements can be realized.

4.2.1. Sensor Data

The transaction access model for the performance data (perishable sensor data) is divided into two distinct groups, "updaters" and "readers". 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 they 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" [Bern87]. 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 locking is necessary for the performance data.

For the accounting and fault information (the persistent sensor data), the updaters from the network monitors append records to existing relations. 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 has to be updated to reflect this fact. Due to the concurrent reading and updating, locking is necessary for the accounting and fault data relations.

4.2.2. Structural Data

The transaction access model for structural data is that it can be both read and written by the network operator(s) or by MIB control processes. Since it is possible that multiple processes may access the same structural data simultaneously, concurrency control has to be implemented. However, since the structural data is updated only very rarely, concurrency control is not a major performance issue with respect to transactions having to block while accessing 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 use the following solution in MANDATE: 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 the structural data. For any other value of StructuralCC, newly-arriving transactions have to follow the 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 structural data are aborted and restarted, thus ensuring that they too follow the locking protocol.

4.2.3. Control Data

The transaction access model for control data is that it can be both read and written by the network operator or by MIB control processes. 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 re-executing 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 is 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. At any given time, no more than one process can update a given set of control variables. Therefore, concurrency control is not required. 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).

4.3. 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 Figure 4.1. 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 sub-networks, 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 sub-network is superposed 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 simulation and optimization algorithms.

There is no 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 that it should 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 indicated by the sensor data.

5. MEETING MIB FUNCTIONALITY GOALS

In the previous section, we have shown how the MANDATE design uses the special characteristics of network management data and transactions to derive improved performance. We follow up in this section by describing how MANDATE uses recent advances in database technology to achieve some of the required MIB functionalities.

5.1. Architectural Issues

Although a 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 the 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 Figure 5.1, 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 that have a full DBMS functionality for cacheing views (subsets) of the MIB data. 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 DBMSs 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 Figure 5.1.

A second server is shown in Figure 5.1 which is used to mirror the database of the main server and thereby provide fault tolerance. All 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.

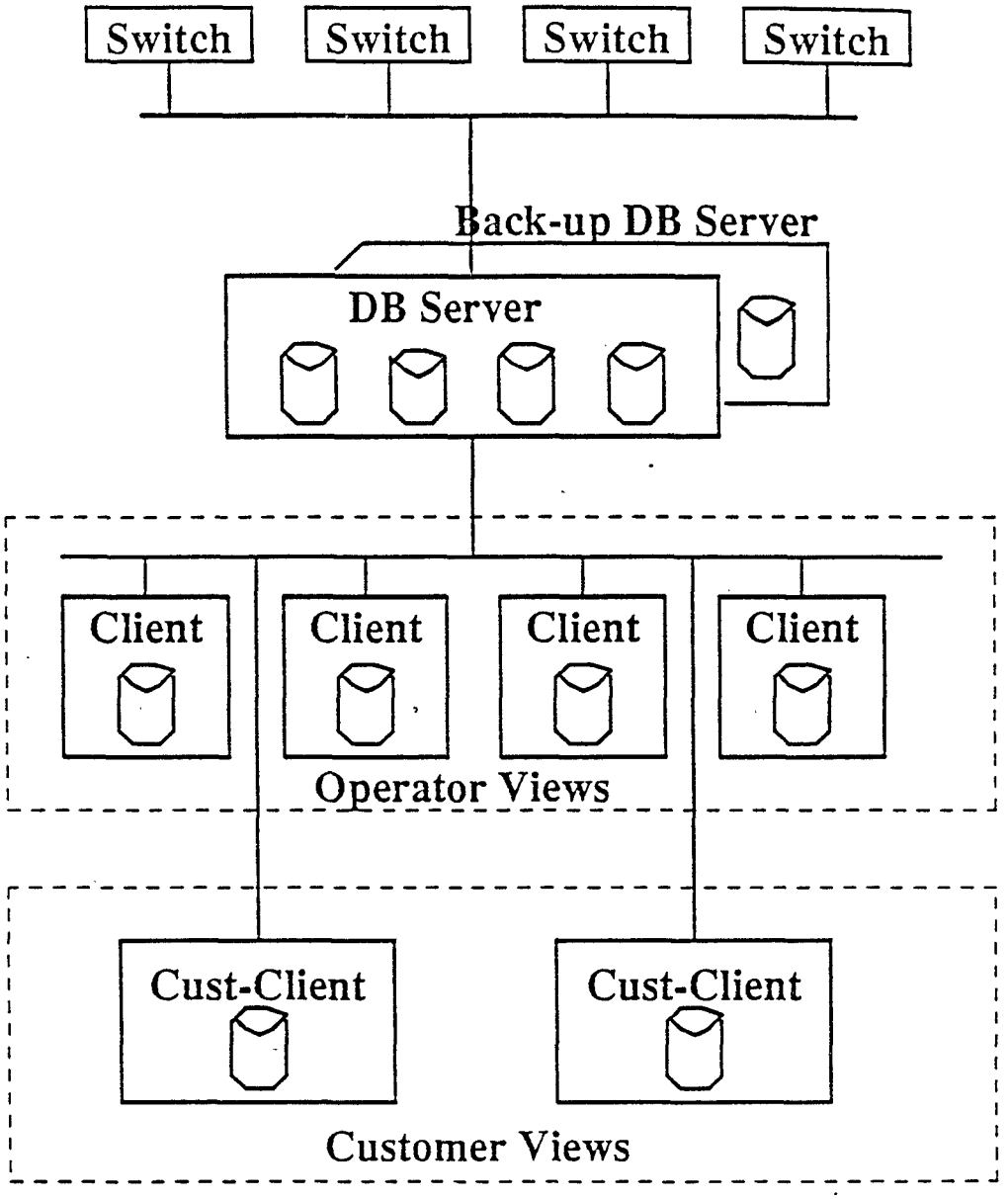


Figure 5.1- An Enhanced Client-Server Database Architecture for MIB

While the above client-server architecture appears on the surface to be an attractive design choice, there are serious problems that may arise in an actual implementation: First, the activities of synchronizing and refreshing downloaded data are repeated very often in a network and therefore, as the number of sites increases, the processing throughput rapidly deteriorates due to blocking. Second, because sites collectively carry a large variety of data subsets, each subset pertinent to a client's function, place of deployment, response requirements, etc., the database servers waste most of their capacity in keeping track of who-needs-what-when in order to propagate changes that affect the clients. On the other hand, if all changes are broadcast to all clients, the clients would be incapacitated due to having to check all updates received, whether or not they were relevant (most of them would be 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 incremental logs maintained at the servers.

Due to the above problems, a straightforward implementation of the client-server architecture is not suitable for network MIBs as it requires enormous processing and data transmission. For example, if queries used to generate and cache the data on the clients is re-executed on the server every time a client needs to refresh its cache, and if the data is transmitted in its entirety, 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. In the following subsections, we show how the problems can be resolved by enhancing the basic client server architecture.

5.2. Incremental Computation Models

Here we present the recently developed concept of *incremental computation models* [Rous86]. Conventional computation models are based on *re-execution,* 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 [Rous91]. 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 re-executing the computation on the whole input.

The following example illustrates the above 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

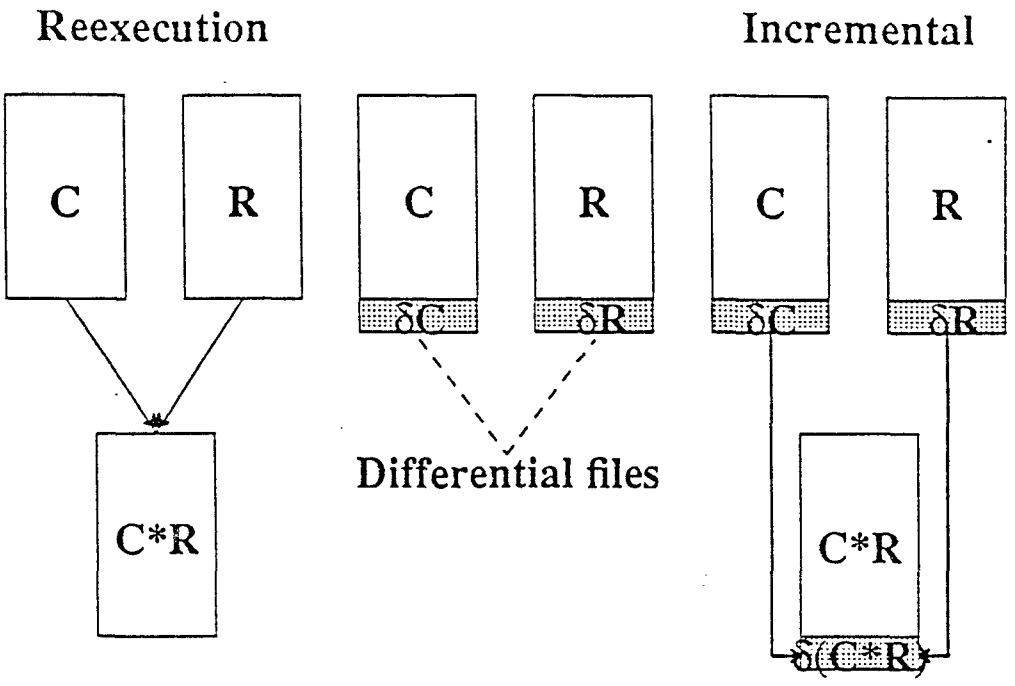


Figure 5.2- Incremental Computation of a Join

$\delta(C*R) = \delta C*R \cup C*\delta R$.

the incremental model, the C*R join is computed only once, as illustrated in Figure 5.2, and thereafter maintained incrementally by using only the differential files δC and δR, $\delta(C*R) = \delta C*R \cup C*\delta R$ (here C and R refer to the updated versions of the relations).

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 life-cycle of the computed information, a concept that is absolutely orthogonal to the re-execution model of transient and non-persistent software, which is the norm. Since computation is done on increments, performance is improved by several orders of magnitude.

5.3. Incremental Client-Server Architecture

The earlier-mentioned problems associated with distributed network database processing in client-server architectures can be resolved by integrating the client-server architecture with the incremental computation model. 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 [Deli92]. More importantly, the incremental model seems especially appropriate for the 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 uptodate 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, the client-server database architecture is combined with incremental computation models to provide the following:

(1) Real-time refreshing of network views.

(2) Immediate update propagation from the primary site to its secondary ones; this guarantees that, in case of failure, the secondary site is uptodate for promotion to be the primary site.

(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 servers I/O.

(5) Preservation of the appropriate level of centralized control.

In summary, the incremental client-server architecture provides the functional advantages of a distributed architecture while retaining the performance of a centralized system.

5.4. Network Control

The fundamental goal of network management is to be able to control the state of the network. In the context of a 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. The continued operation of the network in the presence of faults is achieved through the activation of control processes which are triggered when the network is confronted with unusual circumstances.

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

5.4.1. Local Control

Local control mechanisms rely on local data collection and local decision models. By local we refer to specific 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 [Ephr89]. In this protocol, incoming packets are accepted as long as the number of unacknowledged packets is below a pre-specified 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. In MANDATE, these local control processes are implemented as simple triggers or rules in the MIB.

5.4.2. Global Control

Global control processes rely on network-wide data and 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. Global controls are

usually implemented as algorithms that accept input from the MIB server, perform large computations and return the results to the MIB server.

It should be noted that a significant amount of research has been devoted to the development of distributed implementations of network algorithms [Gall77, Tsit86]. These implementations attempt to develop local controls that approach global performance standards.

5.4.3. 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 allocates incoming calls between a source destination pair equally among three different routes. If one of these routes becomes overloaded, an automatic control process embedded in the switch is triggered such that only twenty percent of all new traffic is sent along the saturated route and the remainder is split equally among the two other routes. Clearly, implementing automatic controls require active database features.

5.4.4. 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 actively through an alarm system that notifies the network manager of network conditions that require actions on his or her part.

The MANDATE design provides passive support for manual controls. Specifically, it provides the operator with

(1) Flexible, fast access to network information.

(2) Embedded optimization and simulation algorithms, described in the next section, which support control decisions.

(3) Facilities that implement specified control decisions.

MANDATE provides active support for automatic control by

(1) Supporting active database elements that can monitor sensor data and automatically set local controls.

(2) Providing embedded optimization algorithms that analyze global network data and automatically propagate control settings throughout the network.

5.5. Embedded Optimization and Simulation Algorithms

The presence of embedded optimization and simulation algorithms as well as the provision of flexible, real-time access to them is a distinct advantage of the MANDATE approach. While some present-day networks employ optimization algorithms to determine global control settings, MANDATE allows for the generation of alternate versions of control settings as well as their detailed analysis using simulation. These features should result in much higher quality control decisions on the part of network operators.

This section deals with the interaction between embedded optimization and simulation algorithms and the MIB. It is important to note that the network control processes which were discussed in the previous sub-section 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 simulation algorithms that output projected network performance based on the control settings, the selection of the final settings and the final execution of these settings. Thus, network control processes are complex mechanisms of which optimization and simulation algorithms are only components. However, optimization algorithms are an integral component of the global controller and therefore require to be fully investigated.

Figure 5.3 below 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, access controls, etc. 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 archived 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 routines which may find commonality between the current network state and some old state and may be able to use old settings. The output from an optimization algorithm will be used to update either an *active* setting or a *non-active* setting. Active settings invoke immediate execution of control action – in such a case, the algorithm would be part of an automatic control process. Implementation of active settings may be achieved through triggers or rules. Non active settings are usually examined by users who make the final decision on implementation. That is, if the user wishes to use the control setting, he or she would transfer it from

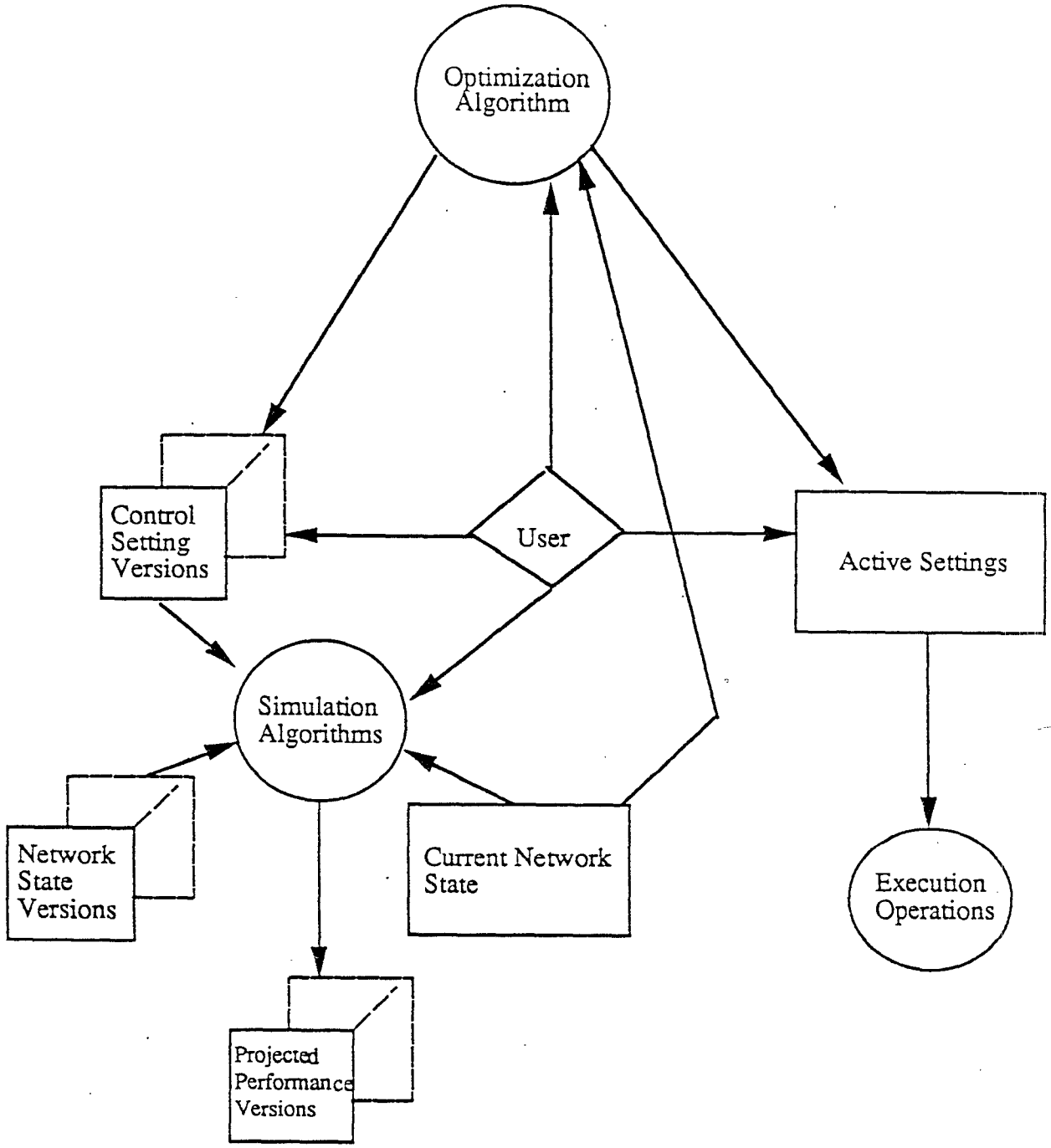


Fig 5.3: Interaction between algorithms and the MIB

the non-active state to the active state. Whether the output of an algorithm is active or non-active is determined by the network state that invoked the execution of the algorithm. For example a case of emergency may trigger the algorithm in an active output mode while a less critical situation may permit non active output and the subsequent examination of the proposed control setting by the operator. In the latter case, the user 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 simulation is to evaluate a proposed central setting (one or more of the versions). The inputs to the simulation 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.

The general environment that must be supported by the system is one in which a user can invoke a given optimization algorithm several times or can invoke alternate algorithms in order to produce different versions of the output (proposed control settings). The user would typically examine and possibly modify various versions and also may wish to invoke a simulation algorithm that produced a detailed analysis of the proposed control setting. Finally, the user would choose one of the versions as a "final version" which in our framework would correspond to making the central setting active.

5.6. 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 post-mortem 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 it 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 is electronically arriving into the system in a continuous fashion [Schw90]. 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 timely 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.

Here we discuss how MANDATE provides temporal views to the network operators, and how the network state information is installed in the database in real-time.

5.6.1. 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 [Sno85].) 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 k-th 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, then 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 $t-Max(P_i)$ 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.

5.6.2. 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 is given by MANDATE to processing these updates as compared to processing the periodic updates. In addition, MANDATE uses sufficient resources and earliest deadline scheduling [Liu73] to ensure that these sporadic updates are always registered in the database.

Under normal operations, when the sporadic updates are arriving at a normal rate, MANDATE ensures that all the periodic updates can be entered into the database system. This is done in the following manner: For each periodic sensor, S_i, the system keeps information about its computation requirement C_i and period P_i. The deadline of each of these periodic update transactions is the time of the next triggering, that is, each task must be completed before the next instance of it occurs. If hardware restrictions require the system to use fixed priority scheduling, then in order to ensure that all updates are registered, it is sufficient to use a processor with sufficient speed such that

$$\sum_{i=1}^{N} \frac{C_i}{P_i} \leq N(2^{\frac{1}{N}}-1)$$

where N is the number of periodic sensors, in conjunction with rate-monotonic scheduling (see [Liu73] for details). On the other hand, if dynamic priority scheduling is possible, then the above condition can be relaxed to employing a processor with sufficient speed such that

$$\sum_{i=1}^{N} \frac{C_i}{P_i} \leq 1$$

in conjunction with earliest-deadline scheduling (see [Liu73] for details).

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, MANDATE uses the Adaptive Earliest Deadline scheduling policy described in [Hari91], which has been shown to provide robust good performance under a variety of workloads and system loading conditions.

6. 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 presented the design of MANDATE, a database system which is geared towards effectively supporting the management of large networks. To the best of our knowledge, this is the first paper to present a detailed architectural design of a network MIB. MANDATE provides the network operators and customers with a 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 handled by MANDATE's internal processes.

We showed here that the network management environment is a specialized application area with unique characteristics that can best be taken advantage of by a 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 basically add network-related modifications on top of commercially available general purpose database systems.

The underlying structural framework of MANDATE is a client-server architecture which is enhanced by the use of an incremental computational model. This architecture results in a high-performance interface which provides the functional advantages of a distributed architecture while retaining the performance of a centralized system.

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 was 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.

A rich variety of control structures are provided in the MANDATE system. Both local control mechanisms, which control specific network components, and global control mechanisms, which control network-wide performance, are supported. For each control mechanism, the user can either control it manually or have the MANDATE system control it automatically. Another distinctive feature of the MANDATE design is its support for embedded network simulation algorithms and optimization algorithms, which are essential for deriving high-quality control decisions.

Network operators and customers 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 are maintained on a real-

time basis, an essential feature for viable network management systems. In addition, MANDATE provides the network operators with a facility for obtaining historical views of the database state. These views are extremely useful for post-mortem analysis of the causes for network faults and also to derive long-term performance profiles.

In summary, MANDATE uses 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. As part of our future research, we plan to test and tune MANDATE by implementing it on a "toy" network and to follow this up with a detailed performance study.

REFERENCES

[Ball92] Ball, L., *Cost-Efficient Network Management*, McGraw-Hill, 1992.

[Bapa91] Bapat, S., "OSI Management Information Base Implementation," *Integrated Network Management, II*, eds. I. Krishnan and W. Zimmer, Elsevier Science Publishers B.V. (North-Holland), 1991.

[Bern87] Bernstein, P., Hadzilacos, V., and Goodman, N., *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[Bort88] Borthnic, S., "Take the Job of Communications Management ... Please," *Business Communications Review*, May-June 1988.

[Cher87] Chernick, M., Mill, K., Aronoff, R., and Strauch, J., "A survey of OSI network management standards activities," *Tech. Report NMSIG87/16 ICST-SNA-87-01*, National Bureau of Standards, 1987.

[DEC89] Digital Equipment Corporation, "Enterprise Management Architecture - General Description," EK-DEMAR-GD-001, 1989.

[Deli92] Delis, A., and Roussopoulos, N., "Performance and Scalability of Client-Server Database Architectures",
Proc. of 18th Int. Conf. on Very Large Data Bases , August 1992.

[Ephr89] Ephremides, A., and Verdu, S., "Control and Optimization Methods in Communication Network Problems," *IEEE Trans. on Automatic Control*, 34(9), September 1989.

[Eric89] Ericson, E., Ericson, L., and Minoli, D., "Expert System Applications in Integrated Network Management," Artech House, 1989.

[Eswa76] Eswaran, K., et al, "The Notions of Consistency and Predicate Locks in a Database System," *Comm. of ACM*, 19(11), November 1976.

[Feld89] Feldkhun, L., "Integrated Network Management Systems," *Integrated Network Management, I*, eds. B. Meandzija and J. Westcott, Elsevier Science Publishers B.V. (North-Holland), 1989.

[Feri88] Feridun, M., Leib, M., Nodine, M., and Ong, J., "ANM: Automated Network Management System," *IEEE Network* , 2(2), March 1988.

[Hari91] Haritsa, J., Livny, M., and Carey, M., "Earliest Deadline Scheduling for Real-Time Database Systems", *Proc. of IEEE Real-Time Systems Symposium*, December 1991.

[Held92] Held, G., *Network Management (Techniques, Tools, and Systems)*, John Wiley & Sons Ltd., 1992.

[ISO90] ISO/IEC, *Information Processing Systems - Open Systems Interconnection - Structure of Management Information - Part 1: Management Information Model*, DP 10165-1, September 1990.

[Gall77] Gallager, R., "A minimum delay routing algorithm using distributed computation," *IEEE Trans. on Communications* , COM-23:73-85, 1977.

[Kler88] Klerer, S., "The OSI Management Architecture: an Overview", *IEEE Network*, 2(2), March 1988.

[Liu73] Liu, C. and Layland, J., "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, Jan. 1973.

[Reed83] Reed, D., "Implementing Atomic Actions on Decentralized Data," *ACM Trans. on Computer Systems* , 1(1), February 1983.

[Rous86] Roussopoulos, N., and Kang, H., "Principles and Techniques in the Design of ADMS±," *IEEE Computer*, 19(12), December 1986.

[Rous91] Roussopoulos, N., "The Incremental Access Method of ViewCache: Concept and Cost Analysis," *ACM Trans. on Database Systems* , 16(3), September 1991.

[Rose91] Rose, M., "The Simple Book – An Introduction to Management of TCP/IP - based internets", Prentice Hall, 1991.

[Schw90] Schwab, B., Wasson, L., and Sholberg, J., "Database Management for an Integrated Network Management System," *Network Management and Control*, ed. A. Kershenbaum et al, Plenum Press, New York, 1990.

[Siro87] Sirovica, D., and Malcolm, J., "Recnik: A Virtual Database for the Coordination of Distributed Network Management Services," *Issues in LAN Management, Proc. of IFIP TC66/W6.4A Workshop on LAN Management*, eds. I. Dallas and E. Spratt, North-Holland, July 1987.

[Snod85] Snodgrass, R., and Ahn, I., "A Taxonomy of Time in Databases," *Proc. of ACM SIGMOD*, June 1985.

[Spri92] SPRINT Network Management Center, Virginia, *Site Visit*, April 1992.

[Terp92] Terplan, K., *Communications Networks Management*, Prentice-Hall, 1992.

[Tsit86] Tsitsiklis, J., and Bertsekas, D., "Distributed asynchronous optimal routing in data networks," *IEEE Trans. on Automatic Control* , AC-31:325-32, 1986.

[Vald86] Valduriez, P., Khoshafian, S., and Copeland, G., "Implementation Techniques of Complex Objects", computation," *Proc. of Int. Conf. on Very Large Databases*, August 1977.

[Valt91] Valta, R., "Design concepts for a Global Network Management Database," *Integrated Network Management, II* , eds. I. Krishnan and W. Zimmer, Elsevier Science Publishers B.V. (North-Holland), 1991.