Design of the MANDATE MIB

Jayant Haritsa, Michael Ball, Nicholas Roussopoulos, John Baras, and Anindya Datta

Institute for Systems Research, University of Maryland, College Park, MD 20742

Abstract

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, a database system that is designed to effectively support the management of large enterprise networks. MANDATE proposes to use 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 that are connected by a communications network. The enterprise communication networks of the future are expected to be heterogeneous in several dimensions: mixed-media physical transmission facilities (e.g. copper, fiber-optic, satellite); multi-media information transmission (e.g. video, voice, data); multiple vendor sub-networks; and varied performance objectives (e.g. high throughput for data, low blocking probability for voice). As a result of this multi-faceted heterogeneity, 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 among management systems for individual subnetworks, efficient management of enterprise networks is an extremely challenging problem.

Network researchers are generally in common agreement that a global network database, which contains all management-related data, is central to the development of an efficient network management system (e.g. [Bapa91, Valt91, Terp92]). This database, referred to in OSI parlance as Management Information Base (MIB), is required to store information on network and system configurations, current and historic performance, trouble logs, security codes, accounting information, etc. [Kler88]. 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 (MAnaging Networks using DAtabase TEchnology), an MIB system that is designed to effectively support the management of large enterprise networks.

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 proposed MANDATE system 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, MANDATE plans to use special characteristics of network management data and transactions, together with recent advances in database technology, to efficiently derive its functionality.

2. ROLE OF DATABASES IN NETWORK MANAGEMENT

The ISO/ANSI standards committee [Cher87] 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 sub-systems. 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 providing for its isolation 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. This database, or MIB, is the conceptual repository of all management-related information.

2.1. Requirements on MIB

Ideally, the MIB module of an enterprise network management system should provide the following major functionalities (a more comprehensive list is described in [Hari92]):

(1) Homogeneous interface: Present a uniform interface to the operator that is independent of the individual sub-network characteristics.

(2) Fault-tolerance: Operate 24 hours on-line since the MIB is the core of the network management system.

(3) Real-time Response: Store and process in real-time the "network health" data which is continuously gathered by monitoring tools.

(4) Decision Support: Answer "what-if" questions by executing on-line simulations, thus helping the operator to evaluate the potential impacts of different control decisions.

(5) Embedded Optimization: Efficiently execute on-line optimization algorithms to adapt the network routing, configuration, etc. in response to changes in the network traffic or connectivity.

(6) 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.

From this list, we observe that an MIB has architectural requirements, interface requirements, temporal requirements, control requirements, and performance goals. Clearly, the design of the MIB is key to providing all of these complex functionalities in an integrated fashion.

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. However, this is not the case for several reasons: First, standard off-the-shelf DBMSs lack many of the required MIB functionalities such as real-time capabilities and decision support facilities. Second, conventional

DBMSs 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. Finally, the objective of conventional DBMSs is to efficiently implement a transaction model that provides the so-called **ACID** property, that is, atomicity, consistency, isolation, and durability. However, this transaction model is unsuitable for processing of network management data, since some of these properties may not be essential here, as explained in the following sections. 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 introduce the design of MANDATE, an MIB that is tuned to the task of network management.

3. RELATED WORK

Issues similar to those addressed in this paper were considered in [Schw90]. The focus in that work was on evaluating how conventional 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 database system whose design is tailor-made for network management. The NETMATE project at Columbia University [Dupu91] has also investigated many of the issues discussed here. They have concentrated on the model of the network and its architectural relationship with management tools, whereas our attention is primarily on the MIB design aspect.

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 architectural design for achieving these functionalities.

4. 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 Figure 4.1, which describes a high-level abstraction of the MIB data model. As explained 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. 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, 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 usage and operational quality. In current large networks, the quantity of sensor data that is gathered may be as large as 20 to 30 gigabytes per day [Spri92].

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 recovery mechanisms 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 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 "sacred" since 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 is necessary to retain some *history* of the data values (using a versioning mechanism) for long-term post-mortem performance and fault analysis.

4.2. Structural Data

In contrast to sensor data, structural data is composed of static or 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.

In a typical large network, the quantity of structural data depends on the detail at which the network is represented and may be of the order of several gigabytes. Most of the structural data is stored at system initiation time and 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. 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).

4.3. 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. In addition to the current parameter settings, the control database also stores a library of pre-defined control settings 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.

5. 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 sensor, structural, and control databases.

5.1. Sensor Data

Two distinct groups of transactions, "updaters" and "readers", access the performance data (perishable sensor data). The updaters are the 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 concurrency control 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 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 has to be updated to reflect this fact. Due to the concurrent reading and updating, concurrency control is necessary for the accounting and fault databases.

5.2. Structural Data

Structural data 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 concurrency control manager for each access to a data object given that regulated access is only rarely necessary. In [Hari92], a simple mechanism is described by which this problem is overcome, thereby resulting in the concurrency control protocol being used *only* when update transactions are executing.

5.3. Control Data

Control data can be both read and written by the network operator or by MIB control processes. The process for changing an existing set of control settings is usually initiated by the network operators. 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. 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 at a node (e.g. introduction of a virus), the links going through the node may be automatically shut down pending investigation of the problem by the network operators. 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.

In a properly designed network, no more than one process can 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 this data category. However, the transaction construct is necessary for installing the updates in order to ensure the atomicity of the updates (half-implemented control settings can cause havoc in the network). Changes to the control data trigger off network executors (see Figure 4.1), which are processes that actually implement in the physical network the new control structure that is logically described by the updated control data.

6. Proposed MIB Design

In conventional database systems, an elaborate set of concurrency control and recovery mechanisms are utilized to provide the ACID property. However, as described in the previous sections, in the network management domain, weaker forms of the ACID property may be acceptable for certain data categories. In the MANDATE design, we propose to incorporate mechanisms that permit concurrency control and recovery to be selectively implemented on a data partition basis. By doing so, we expect to realize considerable performance improvements.

The MIB should be able at all times to provide the operator(s) with a view of the current state 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, we propose to generalize this notion 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.

In the remainder of this section, we will describe how MANDATE proposes to use recent advances in database technology to achieve some of the required MIB functionalities. Due to space constraints, we discuss the system design for only a few functionalities here; a more detailed discussion is provided in [Hari92]. In particular, we focus here on the system architectural model and the network control and decision support mechanisms.

6.1. 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 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 6.1, 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.

While the above client-server architecture appears 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 with each subset pertinent to a particular 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 individual clients. On the other hand, if all changes are broadcast to all clients, the clients would be incapacitated due to having to check all received updates (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 logs maintained at the servers.

6.1.1. Incremental Client-Server Architecture

The above problems associated with distributed network database processing in client-server architectures can be resolved by integrating the client-server architecture with the recently developed concept of *incremental computation models* [Rous86]. In this model, 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. A recent study has shown that this "incremental client-server architecture" 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 slowly over time, the incremental model can realize great improvements in performance and minimize the overhead of view maintenance on the other system functions. In summary, the incremental client-server architecture provides the functional advantages of a distributed architecture while retaining the performance of a centralized system.

In MANDATE, we propose to use the incremental client-server database architecture described above to provide the following: (1) Real-time refreshing of network views; (2) Immediate update propagation from the primary server to its secondary backups – this guarantees that, in case of failure, the secondary server is uptodate for promotion to be the primary server; (3) Parallel access of dynamically distributed data on the enhanced clients and significant reduction on the servers I/O; (4) Preservation of the appropriate level of centralized control.

6.2. 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. 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.

6.2.1. 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]. 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. We propose to implement these local control processes in MANDATE with simple triggers or rules.

6.2.2. 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.

6.2.3. Automatic Control

Automatic controls monitor certain network performance characteristics. 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 requires active database features.

6.2.4. 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.

6.2.5. Control Architecture

In MANDATE, we propose to provide 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.

We also propose to provide in MANDATE passive support for manual controls by providing 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.

6.3. Embedded Optimization and Simulation Algorithms

While some present-day networks employ optimization algorithms to determine global control settings, we plan to support in MANDATE the generation of alternate versions of control settings as well as their detailed analysis using simulation. These features are expected to help network operators make higher quality control decisions. Similar facilities are provided in the NETMATE system [Dupu91].

We now discuss the interaction between embedded optimization and simulation algorithms and the MIB. Figure 6.2 gives a detailed view of this interaction. 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 is 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 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 settings by the operator. In the latter case, the user may wish to run the algorithm several times, possibly with different parameters, and compare the outputs. Typically, in this scenario, as a result of multiple invocations of the algorithm, several versions of control settings would be derived leading to the need for a *version control* scheme.

The role of simulation is to evaluate the performance effects of a proposed control setting. 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 we propose to support in the MANDATE 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 MANDATE, would correspond to making the central setting active.

7. CONCLUSIONS

In this paper, we introduced the design of MANDATE, a database system which is geared towards effectively supporting the management of large networks. In the proposed design, MANDATE provides the 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 handled by MANDATE's internal processes.

The underlying structural framework of MANDATE is a client-server architecture which is enhanced by the use of an incremental computational model. This architecture is expected to result in a high-performance interface which provides the functional advantages of a distributed architecture while retaining the performance of a centralized system. Some of the other special features that we plan to incoporate in MANDATE are selective elimination of concurrency control and recovery overheads, variety of local and global control structures, support for embedded network simulation and optimization algorithms, specialized view-based interfaces and historical views.

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. 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

[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.

[Cher87] Chernick, M. et al, "A survey of OSI network management standards activities," *Tech. Report NMSIG87/16 ICST-SNA-87-01*, National Bureau of Standards, 1987.

[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.

[Dupu91] Dupuy, A., et al, "NETMATE: A Network Management Environment", *IEEE Network Magazine*, March 1991.

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

[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.

[Hari92] Haritsa, J. et al, "MANDATE: MAnaging Networks using DAtabase TEchnology," *Technical Report 92-98*, Systems Research Center, Univ. of Maryland, Sept. 1992.

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

[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.

[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.

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

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

[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.