The OSI network management model

Article *in* IEEE Communications Magazine · June 1993

DOI: 10.1109/35.212418 · Source: IEEE Xplore

Yechiam Yemini
Columbia University
121 PUBLICATIONS **4,197** CITATIONS

The OSI Network Management Model

Balancing the responsibilities of OSI's agents and platforms — and their interaction protocols — is complex, but OSI helps by offering functions lacking in Internet's SNMP.

Yechiam Yemini

YECHIAM YEMINI is a member of the Computer Science Department at Columbia University, where he presently serves as the director of the New York State Center of Advanced Technology in computer and information systems.

There were dramatic shifts in the structure and role of networked systems within enterprises in the past decade. From isolated data-processing islands, networked computing systems have grown into complex mission-critical enterprise-wide systems. The network, the computer, and the very enterprise rapidly are becoming indistinguishable. These changes lead to significant risks and cost exposure associated with operations. For example, failures of a bank network can paralyze its operations, delays of security trades through brokerage system bottlenecks can cost in dollars and customers, and loss of hospital lab reports can prevent timely diagnosis and care. The goal of network management technologies is to reduce the risks and costs exposure associated with operations of enterprise systems.

Management systems are responsible to monitor, interpret, and control the network operations. A typical management system is depicted in Fig. 1. Vendors equip their devices with agent software to monitor and collect operational data (e.g., error statistics) into local databases, and/or detect exceptional events (e.g., error rates exceed threshold). Management platform workstations query device data, or obtain event notifications through management protocols. The management platform supports tools to display the data graphically, interpret it, and control operations.

This management paradigm is *platform centered*. Management applications are centralized in platforms, separated from the managed data and control functions in the devices. Platform-centered management reflects older network environments where devices lacked resources to run management software, management data and functions were relatively simple, and network organizations could devote the personnel needed to handle operations. The implications of these assumptions and their validity for current networks will be considered later.

The main challenge of management standardization is to develop conventions to support integrated management of heterogeneous networks. Platform-centered management requires a few standards. First, access by platforms to multivendor devices must be unified through a standard management protocol. Second, the structure of the agent's management databases, manipulated by the protocol, must be standardized. Together, these standards permit a platform to access and manipulate managed information at multivendor device agents. The OSI and Internet management models seek to standardize both areas.

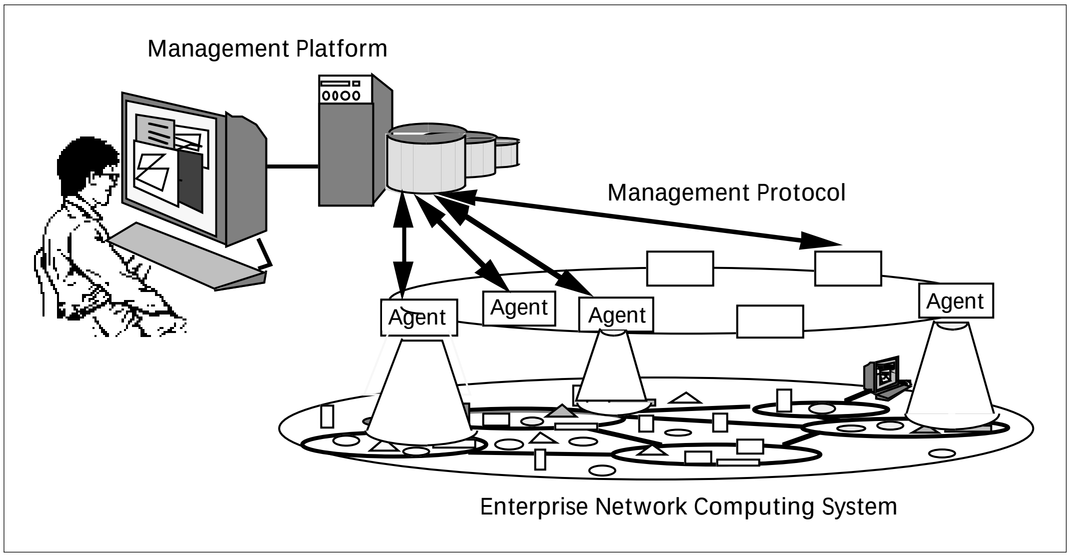
Merely moving management information from devices to platforms, however, is insufficient to eliminate the curse of heterogeneity. Two additional barriers to integrated management arise: platform and semantic heterogeneity.

Platform heterogeneity means that management applications must be replicated for each major platform. For example, a device vendor wishing to offer six applications over five platforms may need to develop and maintain 30 product versions. Therefore, a number of recent consortia (e.g., OSF, XOPEN, POSIX) are pursuing management platform standards.

Semantic heterogeneity arises when different devices use different information to represent similar network behaviors. A management application program requires a uniform semantic model of the managed information it processes. It is necessary to standardize the very meaning of managed information. Various IEEE and CCITT protocol committees pursue this challenge, building managed information standards for protocol entities.

Why Is Management Difficult?

Consider an example of a network "storm" to illustrate management complexities. Storms involving rapid escalation of cascading failures are not uncommon in networks. Figure 2 depicts



■ **Figure 1.** *Architecture of a network management system.*

a T1 link multiplexing a large number of connections (e.g., X.25 virtual circuits, or TCP) to a server/host. Suppose a long burst of noise disrupts the link causing packet loss (Fig. 2a). Logical link level protocols (above the physical layer) invoke automatic retransmission. They result in a burst of retransmission tasks at the interface processor queue (Fig. 2b) loading its queue and leading to its thrashing. Higher layer transport entities time-out and respond with a burst of corrective activities (e.g., reset connections). This burst processing of communications at host CPUs (Fig. 2c) leads to their thrashing, too.

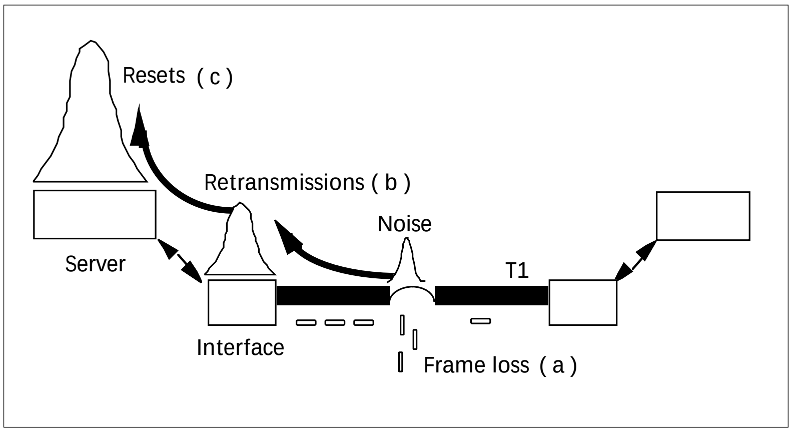
Generally, protocol stack mechanisms handle lower-layer problems through corrective actions at higher layers. These mechanisms can escalate the very problems they intend to solve.

How can such complex network fault behaviors be monitored, detected, and handled? Suppose that relevant operational variables (e.g., T1 bit-error rates and the size of the interface processor queue) can be observed as depicted in Fig. 3. The storm formation could be detected from the correlation of the sudden growth in error rates and the resulting growth in queue size.

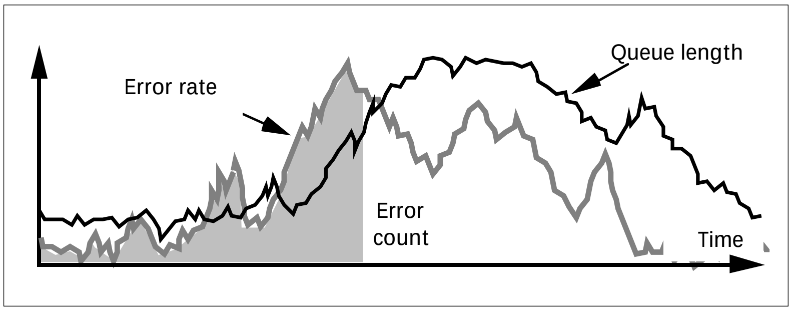
What management information should be used to capture these behaviors? The Simple Network Management Protocol (SNMP) uses a simple model for the structure of managed information (SMI) [2] involving six application-defined data types and three generic ones.

Temporal behaviors are described in terms of counters and gauges. An error counter represents the cumulative errors (integral) since device booting (the area under the error rate curve in Fig. 3). A gauge can model the queue length. The values of these managed variables can be recorded in an agent's management information base (MIB) [3], where they can be polled by a platform. An error counter, however, is not useful for detecting rapid changes in error rates to identify a storm. A platform must sample the counter frequently to estimate its second derivative, leading to unrealistic polling rates.

OSI management uses an object-oriented model of managed information [9, 10]. The behaviors of interest: noise, errors, and queue length are different forms of a time series. A generic managed



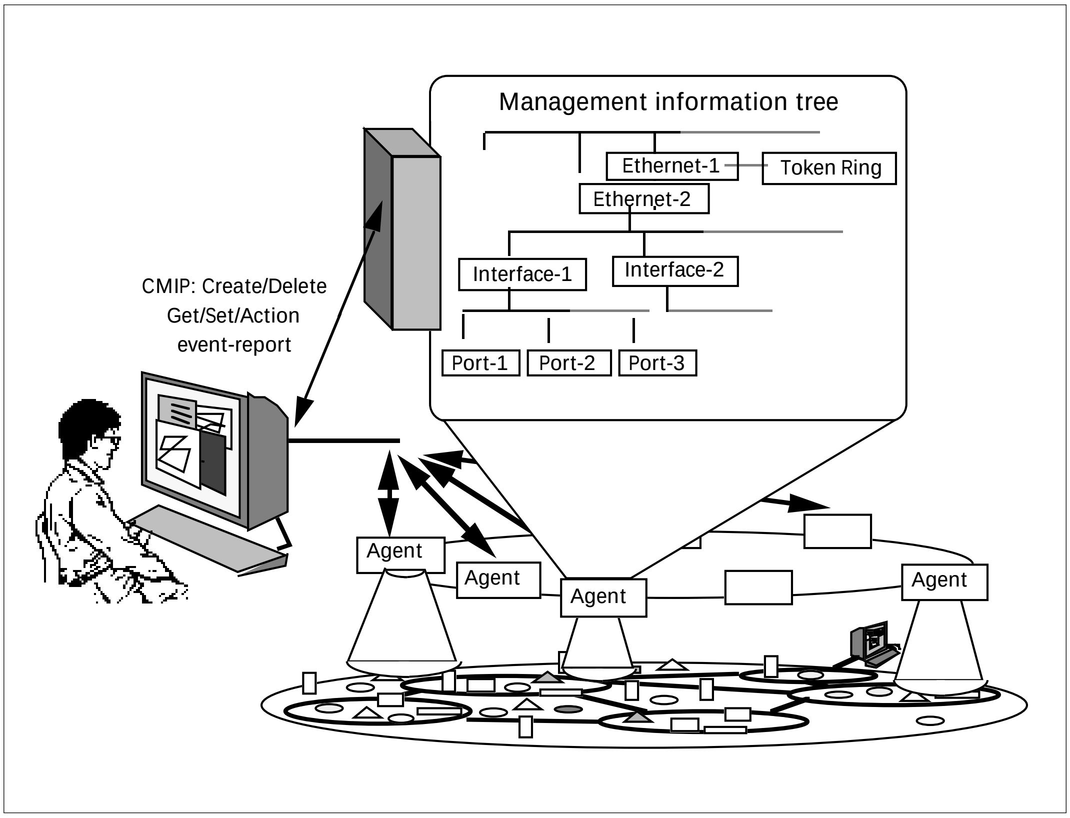
■ **Figure 2.** *Formation of a network storm.*



■ **Figure 3.** *Temporal behaviors correlation.*

object (MO) class may be defined to describe a general time series. This MO may include data attributes of the time series and operations (methods, actions) to compute functions of the time series (e.g., derivatives). This MO also may provide generic events notifications (e.g., whenever some function of time series exceeds threshold). The generic time series MO class may be specialized to define MO subclasses to model the bit-error rate of the T1 link and the queue length of the interface processor. A management platform can create instances of these MOs, within device agents' databases. The device agent can monitor the respective network behaviors and record the respective

OSI management communications require connection-oriented transport and rely on the OSI application layer environment.



■ **Figure 4.** *Overall architecture of an OSI management system.*

values in these MO instances. Furthermore, the platform can enroll with the agent to receive notifications of events describing rapid changes of error rates and excessive processor queue.

To identify a storm it is necessary to detect not only error rate and interface queue threshold events, but to identify correlation among them. Unfortunately, the observation processes used for event detection may result in decorrelation of behaviors. Threshold excesses must be sustained over sufficient window to result in event notification and avoid spurious alerts. Implementation may use different sampling rates and detection windows for the error rate and queue length. Thus, either of the events may be detected singly, or the events may be detected in inverted temporal order. This decorrelating effect of observations can lead to erroneous interpretation. Often, hundreds or thousands of alerts may be generated by a fault. These events must be correlated to detect the problem's source. The results of such analysis may be very sensitive to the choices of managed information provided by devices and the implementation details of monitoring processes.

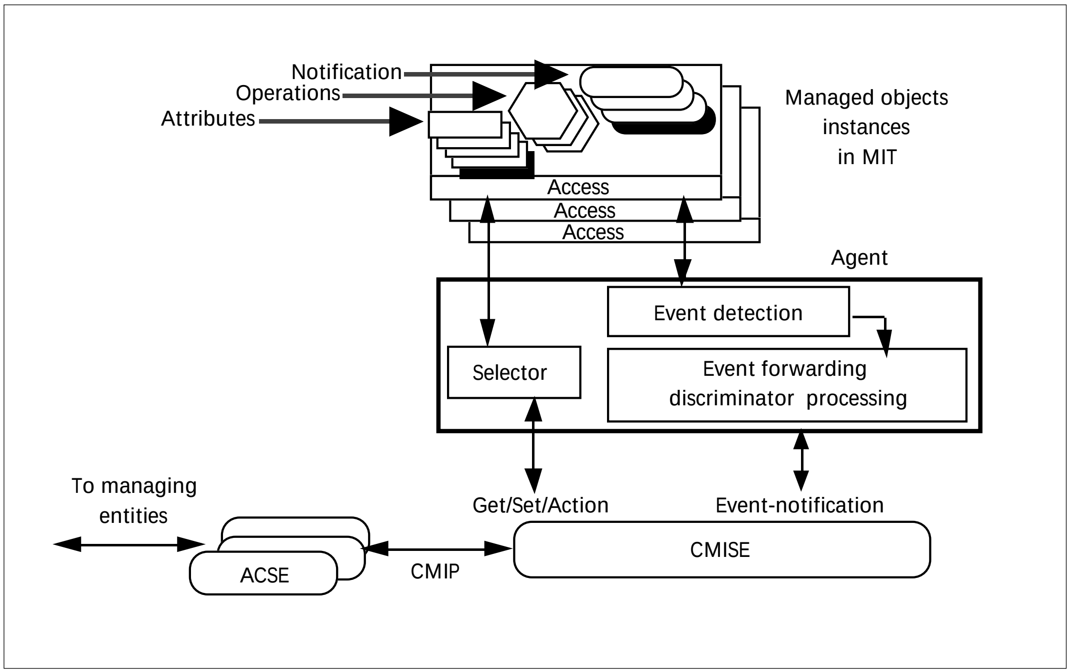
To make things worse, the devices involved in the storm formation are typically manufactured by different vendors. The managed data and its meaning may vary greatly among devices, rendering interpretation difficult. Moreover, networks often are operated by multiple organizations, each responsible for a different domain. Faults escalating across such domain boundaries may be particularly difficult to detect and handle. In the example, the T1 links may be managed by a telephone company while the layers above are operated by a user organization. The user may not be able to observe the behavior of the T1 layer and the telephone company may not be able to observe the behaviors of higher layers.

OSI Management Model Overview

The OSI management model [5] is depicted in Fig. 4. The managing platform on the left uses the common management information protocol (CMIP) to access managed information, provided by an agent residing in a LAN hub on the right. The agent maintains a management information tree (MIT) database. The MIT models a hub using MOs to represent LANs, interfaces, and ports. A platform can use CMIP to create, delete, retrieve, or change MOs in the MIT; invoke actions; or receive event notifications.

The MIT contains instances of MOs organized on a hierarchical database tree, similar to the X.500 directory tree [12]. An MO instance includes attributes that serve as its relative distinguishing name (RDN). The RDN attributes uniquely identify the instance among the siblings of its MIT parent. In the hub example of Fig. 4, a port identification number may be used as the RDN to identify ports of a given interface MO. By concatenating RDNs along the MIT path from the root to a given node, a unique distinguishing name (DN) is obtained. This DN is used by CMIP to identify a node and access its managed information.

In contrast to SNMP MIB, the MIT is a dynamic database. SNMP also uses a tree [3] to store managed information. However, the structure of the MIB is static and is determined at its design time. CMIP provides CREATE/DELETE primitives to change the MIT dynamically. A dynamic database



■ **Figure 5.** *Managed information communication architecture.*

can provide flexibility and efficiency in managed information access. Managing entities can control the contents and structure of the database. The database also may be flexibly organized to reflect specific device configurations. A static database structure may lead to difficulties in handling composite device structures. Different components may require their own database models. However, they cannot be unified into a single MIB due to its static structure. Therefore, complex hubs often include multiple SNMP agents (each handling a different component). A dynamic MIT permits these different database models to be easily unified.

A dynamic management database, however, presents significant implementation complexities. The resources required to store and process managed information cannot be predicted at design time. Managers may extend the MIT beyond available agent's resources. Changes in the MIT may result in corruption of the database. For example, an MO may be deleted while other MOs contain relationship pointers to it. Application software designers cannot share a single model of the MIT contents, as each application needs to build and maintain its own MIT subset.

Management Communication Model

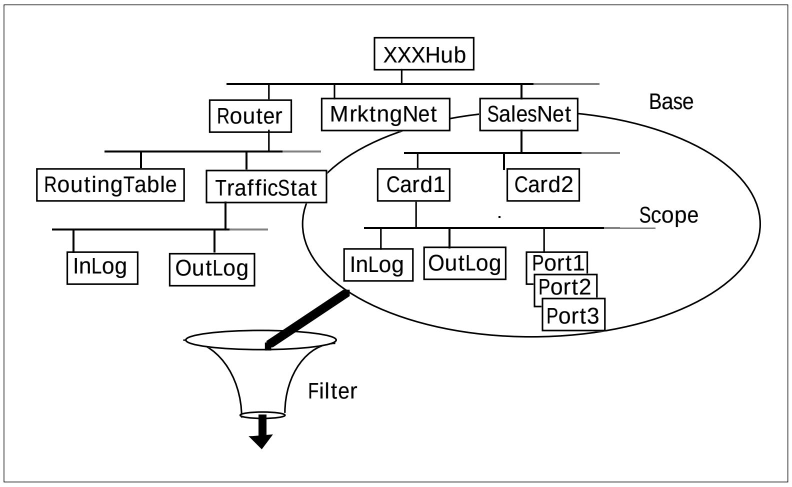
OSI management communications require connection-oriented transport and rely on the OSI application layer environment. (Consult Reference [12] for OSI application layer details.) Agents (managed entities) and managers (managing entities) are viewed as peer applications that use the services of a common management information service element (CMISE) to exchange managed information [6]. CMISE provides service access points (SAPs) to support controlled associations between managers and agents. Associations are used to exchange managed information

Management communication services
M-INITIALIZE: Establish management association M-TERMINATE: Terminate management association M-ABORT: Unconfirmed termination
Management information tree operations
M-CREATE: Creates an MO instance record in the MIT M-DELETE: Deletes an MO instance from MIT
Managed information manipulation services
M-GET: Retrieve information M-CANCEL-GET: Cancel retrievals M-SET: Change an attribute value M-ACTION: Invoke an MO operation M-EVENT-REPORT: Generate an MO event report to a manager

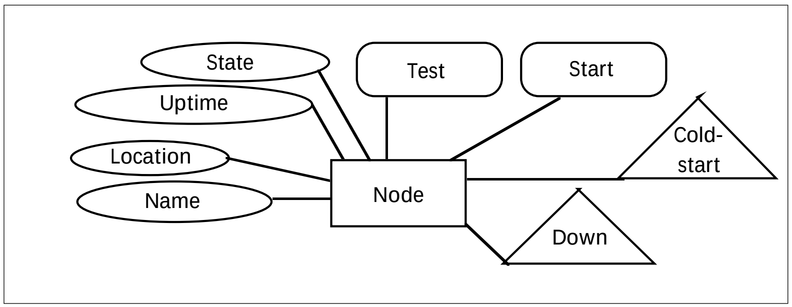
■ **Table 1.** *SAPs provided by a CMISE entity.*

queries and responses, handle event notifications, and provide remote invocations of MO operations. CMISE utilizes the services of OSI's association control service element (ACSE) and the remote operations service element (ROSE) to support these services [12]. A typical structure of an agent communication environment is shown in Fig. 5. A symmetric organization governs the structure of peer managing entities.

The top section of Fig. 5 describes the structure of the MIT. MO instances and their attributes, operations, and event notifications are depicted as shaded rectangles at the top. The OSI agent provides selection functions to locate the MO records accessed by Get/Set/Action SAPs of CMISE. The agent also provides event detection and forwarding of notifications to managing entities enrolled (through MIT records) to receive them. A CMISE entity provides SAPs (depicted in Table 1) to support communications with the agent. It dis-



■ **Figure 6.** *Aggregated and selective retrieval.*



■ **Figure 7.** *Example of an MO.*

patches/receives CMIP PDUs to/from other service elements such as ACSE and ROSE. These PDUs are exchanges through a connection-oriented transport. CMIP PDUs are best viewed as carriers of requests and replies generated by respective CMISE primitives. For example, a CMISE M-GET accessed by a manager generates a CMIP GET-REQUEST PDU to the agent and respective GET-RESPONSE PDUs from the agent.

The interactions pursued by management applications peers are typically confirmed through the standard OSI request-reply model. For example, an invocation by a manager of the M-INITIALIZE SAP results in a CMISE invocation of the ACSE through a CMIP PDU. The manager ACSE sends an association request to a peer. The peer ACSE at the agent passes the CMIP request to the agent's CMISE. A confirmation PDU will then propagate back from the agent's CMISE through an ACSE and back to the originating manager.

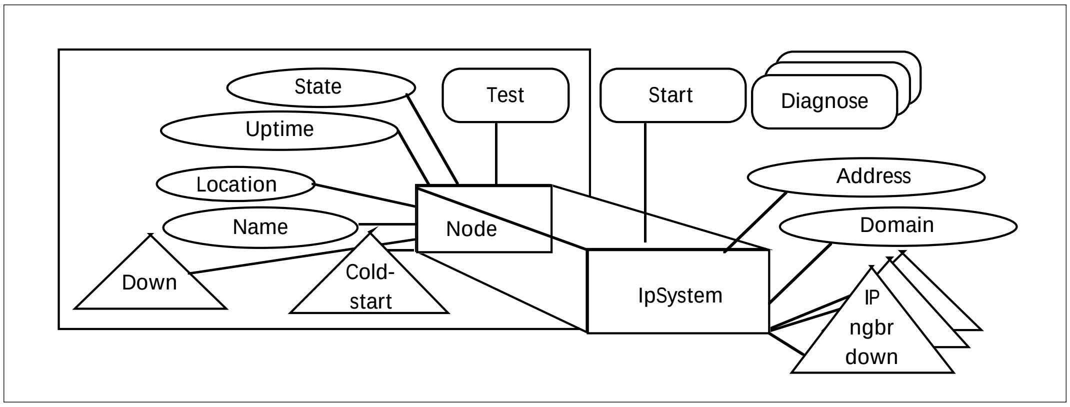
The core services of CMISE provide access to managed information. The GET construct provides means for bulk retrieval and agent's information filtering. This is illustrated in Fig. 6. To accomplish bulk retrieval, a GET need only specify a subtree of the MIT from which data is to be retrieved. This subtree is specified by its base node and the scope of the GET request. To specify a selection criterion, a GET must provide a filter defined by a simple language. Retrieval of all port data on SalesNet whose error rates exceed some threshold is illustrated in Fig. 6. The GET request identifies the scope of the search and filter and the agent performs the search and selection.

Remote invocation of operations is accomplished through M-ACTION. It is necessary to specify the MO instance, the action to be invoked, and the parameters to be passed to it. The invocation is supported through the ROSE. Event notifications are handled by enrolling appropriate records on the MIT, using M-CREATE. A manager uses M-CREATE to place an event-notification-managed object on the MIT. Upon detection of an event, the agent uses the MIT to identify subscribers for notifications. An M-EVENT-NOTIFICATION is generated for each such subscriber.

SMI Model

The structure of managed information (SMI) model plays a central role in the OSI standard. It is introduced in [8] and is elaborated in the guidelines for the definitions of managed objects (GDMO) [10]. This model is based on an extended object-oriented (OO) data model [16]. MOs, like OO classes, provide templates to encapsulate data and management operations (methods, actions) associated with managed entities. MO extends the class concept to include event notifications. Event notifications add a new dimension. Traditional OO software assumes a synchronous model of interaction between an object and its users (programs). Programs may invoke methods synchronously. On the other hand, events may occur independently and asynchronously with the manager computations that access them.

The MO model supports inheritance. An MO definition can include attributes, operations, and events of a more general MO. For example, a general MO describing an interface may be used to define specialized interfaces (e.g., Ethernet, Token-ring). The data, operations, and events associated



■ **Figure 8.** *A subclass of node.*

```
<class-label> MANAGED OBJECT CLASS
[DERIVED FROM <class-label> [,<class-label>]*;]
[ALLOMORPHIC SET  <class-label>  [,<class-label>]*;]
[CHARACTERIZED BY  <package-label>  [,<package-label>]*;]
[CONDITIONAL PACKAGES  <package-label>  PRESENT IF <condition-definitions>
       [,<package-label>PRESENT IF <condition-definitions>]*;]
[PARAMETERS <parameter-label> [, <parameter-label>]*;]
REGISTERED AS <object-identifier> ;
```

■ **Figure 9.** *Sample MO class template.*

```
eventLogRecord MANAGED OBJECT CLASS
DERIVED FROM logRecord;
CHARACTERIZED BY  eventLogRecordPackage PACKAGE
     ATTRIBUTES managedObjectClass GET, managedObjectInstance GET;;;
CONDITIONAL PACKAGES
     eventTimePkg PACKAGE ATTRIBUTES eventTime GET;;
     PRESENT IF the event time parameter was present in the CMIP event report;
REGISTERED AS {smi2MobjectClass 5};
```

■ **Figure 10.** *Example of eventLogRecord.*

with an interface MO will be inherited by these specialized subclasses. Inheritance is primarily a syntactic mechanism as one could simply include the definitions of the superclass in the subclass MO definitions to accomplish the same effect. To illustrate inheritance, consider an MO defining a class of node objects as depicted in Fig. 7. Ellipsoidal shapes describe data attributes. Rectangular shapes describe operations to test and start a node. Events are described by triangular shapes.

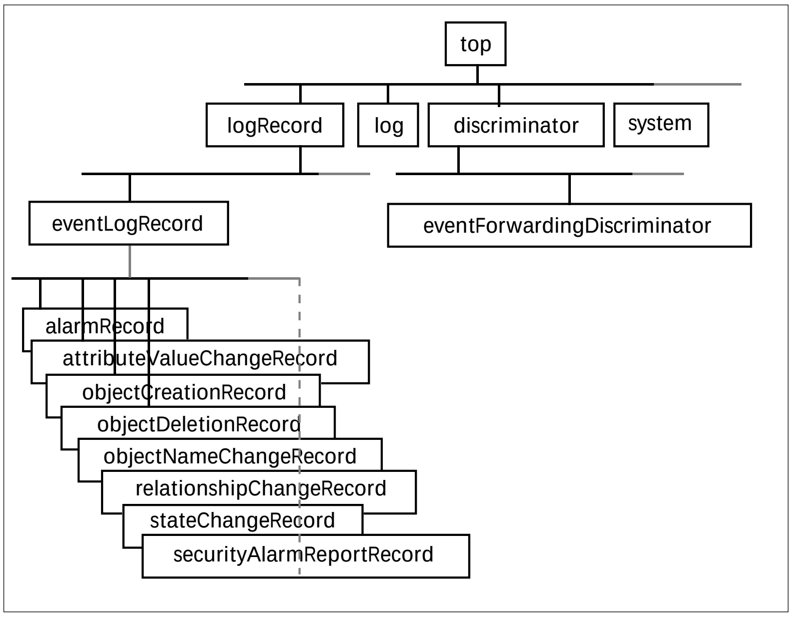
Consider now a specialization of a "node"—an IpSystem MO. An IpSystem can be defined as a subclass of node. It inherits all the node attributes, operations, and events. The IpSystem may replace some of these inherited components (e.g., a new start operation) and add new attributes, operations, and events.

Relationships Are Significant in Management

Relationships among managed data items are of great importance in correlating information. In the earlier storm example, it was necessary to correlate observations of physical layer errors with those of an interface processor queue handling retransmission tasks. It would have been necessary to represent the relations among these objects to be able to correlate their behaviors. The managed information model of OSI, in contrast to SNMP, provides explicit means to represent relationships.

An MO may include relationship attributes with pointers to related MOs. For example, a port attribute representing the relationship "contained-in" may include a pointer to the interface object that contains it. The pointer value is the distinguishing name (DN) path identifier of the interface object on the MIT.

The OSI model includes a number of generic relationships that may be used in modeling MO such as "is-contained-in," "is-peer-of" (for protocol entities), and "is-backup-of" (for systems or components). The use of relationship attributes is similar to techniques used in the network model of databases [17]. It achieves great generality in representing, in principle, any entity-relationship model. For example, one can easily identify and retrieve



■ **Figure 11.** *The MO class hierarchy.*

information associated with all ports contained in a given interface, by traversing the respective relationship pointers. Of course, traversal may require substantial manager-agent interactions to retrieve and dereference pointers. This could have been simplified if the protocol included traversal primitives (GET-NEXT) to follow relationship pointers, similarly to network databases.

GDMO Provides Syntax for MO Definitions

The GDMO introduces substantial extensions of ASN.1 to handle the syntax of managed information definitions. A new language structure (template), is introduced to combine definitions. Templates play a similar role as ASN.1 Macro, except they do not lend themselves to simple extensions of ASN.1 compilers. A sample template is the MO class template (Fig. 9). It is used to define MO structure and register the definitions on the ISO registration tree.

```
system MANAGED OBJECT CLASS
DERIVED FROM  top;
CHARACTERIZED BY  systemPackage PACKAGE
     ATTRIBUTES systemID GET, operationalState GET, usageState GET,
     administrativeState GET REPLACE, managementState GET;
     ATTRIBUTE GROUPSstate, relationship;
     NOTIFICATIONS objectCreation, objectDeletion, objectNameChange,
     attributeValueChange, state Change, ......,environmentalAlarm;;;
CONDITIONAL PACKAGES
     dailyScheduling PRESENT IF both the weekly scheduling and
     external scheduler package  not present  in an instance
     ..........
     repairStatusPkg PACKAGE
     ATTRIBUTES...
     PRESENT IF both the weekly scheduling and external
     scheduler package are not present in an instance
.................................
REGISTERED AS {smi2MOBJECTClass 14};
```

■ **Figure 12.** *Schematic subset of system MO.*

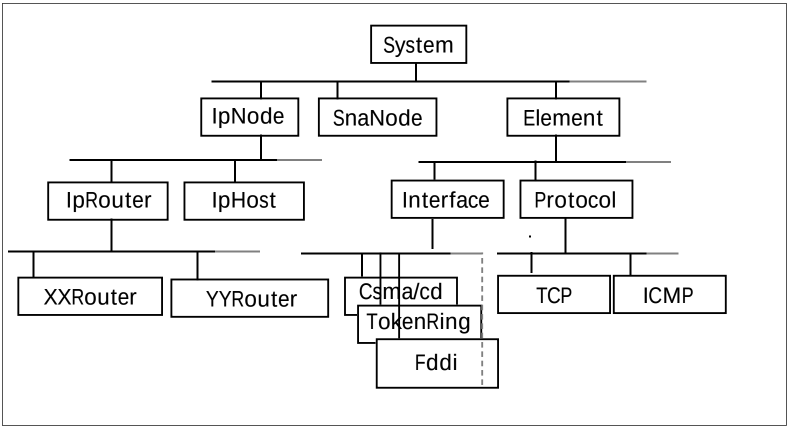
The <class-label> is a place-holder for an MO name. The "Derived from" section describes the superclasses whose definitions are inherited by the MO. The "Characterized by" part includes the body of data attributes, operations, and event notifications encapsulated by the MO. "Packages," "Conditional packages," and "Parameters" are templates used to combine definitions of attributes, operations, and event notifications. The "Registered as" part registers the MO definition on the ISO registration tree.

An example of definition of an eventLogRecord (Fig. 10) [9], using this template, follows. An eventLogRecord inherits attributes of a general logRecord (its superclass). It includes definitions taken from a package for eventLogRecordPackage and a conditional package eventTimePkg, and is registered on the OSI registration tree as the subtree labeled 5 under the label smi2MobjectClass. Attributes are followed with descriptors such as GET/REPLACE, denoting read/write access mode. Notice the informal statement of the condition under which the definitions by the conditional package are to be included. Thus, automated compilation of definitions, unlike SNMP MIBs, may be impossible.

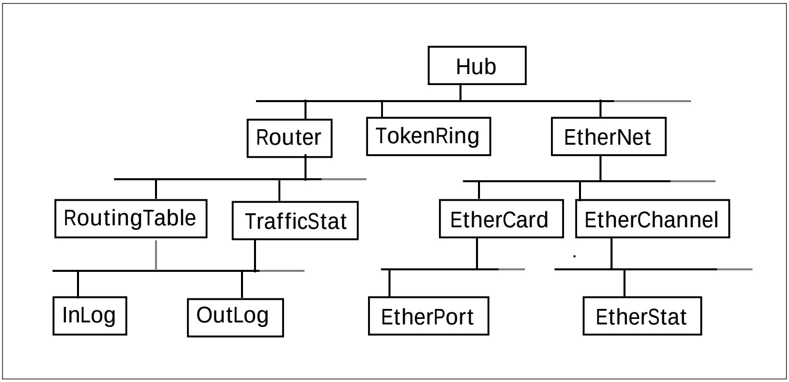
The definitions of managed information (DMI) [9] define a class hierarchy as depicted in Fig. 11. Boxes represent different MO classes, while the tree represents inheritance relationsamong them.

These generic MOs focus on definitions of various forms of management logs. The system MO is the main tool in building MOs associated with a given system. It includes attributes to identify the system, represent its operational and administrative state, and provide generic notifications and packages of definitions for handling scheduled operations maintenance. A schematic subset of its definition is provided in Fig. 12.

For example, consider the problem of developing a class hierarchy to model typical networked systems, as a specialization of the system MO. A possible class hierarchy is illustrated in Fig. 13. This hierarchy considers two kinds of systems: complex systems (as on the left part of the tree) and simple systems, or elements (as on the right side of the tree).



■ **Figure 13.** *An MO class hierarchy to represent networked devices.*



■ **Figure 14.** *Typical hub components and containment.*

Putting It Together

Building an OSI Managed Element

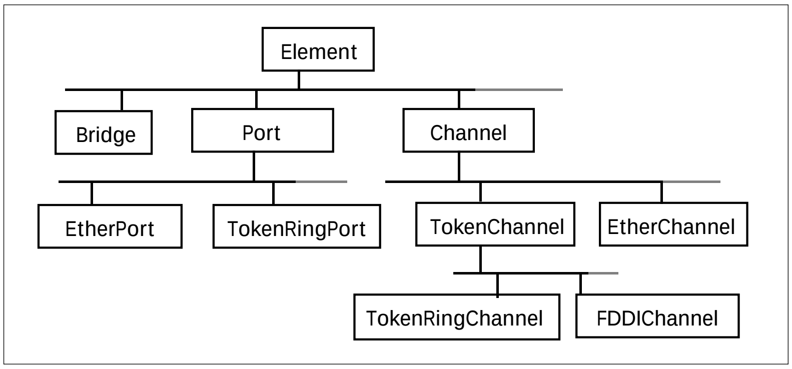
This section completes the picture through a brief sketch of OSI modeling of a LAN hub. Typical hub components and their containment relations are depicted in Fig. 14.

Step 1: Identify Class Structure and Inheritance Relations Among MOs — An MO must be designed for each managed component. The first step is to identify similarities of managed elements and capture them in MO classes, to use inheritance.

A possible class inheritance hierarchy is depicted in Fig. 15. (For example, EtherPort and TokenPort components may share some attributes, operations, and events. The MO describing them may be developed as specialization of a generic port object.)

Step 2: Design and Specify MO Syntactical Structures Using GDMO — Using the GDMO, define managed attributes, operations, and event notifications for each of the MO classes needed. An example of port MO definitions is shown in Fig. 16. MO libraries defined by protocol committees (e.g., FDDI) may be used to capture standardized components.

Step 3: Design Generic MIT Structure for the Device — Design of the MIT follows the containment tree of Fig. 14. Each component is replaced by respective MO instances. For each MO, it is necessary to identify respective attributes forming a unique relative distinguishing name (RDN). For each device component, the respective MO instance must be created on the MIT (using a CMIP Create primitive). Dynamic managed objects (e.g., different logs) may be created and deleted by managers during network running time. Relationship attributes values are set after the respective MO instances are located on the MIT. For example, a "contained-in" relationship may be associated with port-card pairs. An instance of an etherPort MO may include a pointer to the etherCard MO instance. Similarly, an etherCard instance may point to an etherChannel instance (subnet) to represent the relationship "attached-to."



■ **Figure 15.** *A class hierarchy for hub MOs.*

```
port MANAGED OBJECT CLASS
DERIVED FROM  element;
CHARACTERIZED BY  portPackage PACKAGE
       ATTRIBUTES portNum GET, portStatus GET,...;
       OPERATIONS diagnose,disconnect,connect...;
       NOTIFICATIONS portFailure, portInitialized..;
.................................
REGISTERED AS {......};
```

■ **Figure 16.** *Example of port MO definitions.*

Critical Assessment

The OSI model seeks to provide a comprehensive framework for handling management of arbitrarily complex systems. We will briefly evaluate some tradeoffs associated with this generality, and contrast the choices of OSI with those of the Internet SNMP.

Managed Information Model

OSI provides an extended OO database framework to model managed information. It seeks to maximize the information modeling power to handle complex systems. Management information bases, however, need to balance conflicting requirements for functionality and real-time performance under resource constraints. It is not yet established whether the OSI design choices can strike such balance. For example, a dynamic MIT database can easily saturate agent's memory and/or processing resources. Demand for event notifications may tie agent's processing and communication resources, starving GET or ACTION requests (and vice versa).

Generally, the performance of OO databases is not yet understood [16]. Deletion of MO records may result in orphaned relationship pointers, requiring complex garbage collection at agents. Multiple managers pursuing CREATE/DELETE activities can lead to inconsistent views of the MIT.

Complexity of information model can lead to conformance difficulties. For example, the semantic of event notifications must be formally captured to permit specifications of conformance criteria. The meaning of events, however, is tied to device operations extrinsic to the MIT and its contents.

In contrast, SNMP pursues a simple static MIB, seeking to minimize and constrain the agent's resource demand. The memory and processing resources needed to handle the MIB may be carefully evaluated and planned at design time. However, the data modeling power of SNMP is limited. For example, composite systems may need multiple MIB instances to represent their different parts. These MIBs cannot be combined into a single database nor be accessed from a single agent. Thus, a composite system requires as many SNMP agents as its components. As another example, lack of explicit modeling of relationships limits the ability of applications to correlate managed data. Recent proposals of SNMP V.2 [4] seek to resolve some of SNMP's information modeling limitations. The proper balance between modeling power and performance of managed information databases remains an elusive design goal.

Managed Information Access Model

The OSI model introduces two important functionalities missing in SNMP: bulk and selective retrievals. Both capabilities are central in controlling the flow of management information. Without bulk retrieval, managers are forced to pursue a large number of polling requests. Without selective (filtered) retrieval, managers are forced to retrieve large amounts of irrelevant data.

Explicit invocation of agent's operations is another OSI capability missing from SNMP. Remote invocations are important for distribution of management computations to agents and improve manager control over agent's activities. They also can reduce the complexity of manager-agents interactions by limiting it to procedure interfaces. SNMP supports implicit invocations as side effects of SET requests. A diagnostic operation, for example, may be invoked by a "set" of a respective variable. Implicit invocations offer only limited capa-

Management should pursue flexible decentralization of responsibilities to devices and maximal automation of management functions through application software.

bilities in passing parameters and in synchronizing invocations with managers. Additionally, they increase agent's complexity since SET requests must be trapped to invoke respective procedures.

Communication Model

OSI management uses connection-oriented transport and confirmed interactions. These provide reliability and enable bulk retrieval; a single GET can result in multiple-linked replies. They require, however, complex communication environment and result in failure-sensitivity. During network stress time, connections may not be sustainable over sufficiently long time to accomplish the management functions needed. Management entities may need to spend significant time and resources in handling lost connections. Connection-based transport may become an obstacle in accomplishing management interactions at a time when they are needed most.

Conversely, SNMP communications use a connectionless datagram transport (UDP) with confirmed GET/SET interactions and unconfirmed event notifications (TRAPs). The responsibility to ensure reliable communications is passed to agent/manager applications. For example, managers can detect loss of a GET/SET request when the GET-RESPONSE confirmation does not arrive. They can ignore the loss, reissue the request, or choose other alternatives to recovery. During stress time, managers may flexibly adjust their computations to handle loss, rather than confront an all-or-nothing choice of a reliable connection service. A datagram model requires a simple communication environment that is easy to implement. Managers, however, can only retrieve information that fits within a single UDP frame. This limits bulk retrieval mechanisms.

The Platform-centered Management Paradigm

How useful is the OSI model in supporting platform-centered management? The OSI model prescribes powerful agents requiring substantial computational resources, on par with the resources available at the platform. This raises interesting questions concerning allocation of responsibilities among platforms and agents. If agents are to be as powerful, what functions should be removed from them and assigned to platform managers and why? If the platform is to play an incidental role in management, does it require a comprehensive, or even any, general management protocol? Why should it not limit exchanges with agents to application-specific APIs (e.g., using ROSE or an RPC)? How does a maximal access model, which exposes internal object details of managed entities to a platform, serve a minimal platform? However, if the platform is to be maximal, why are maximal agents required? The balance among the complexities and responsibilities of agents, platforms, and their interaction protocols is not yet understood.

How useful is platform-centered management? Platform-centered management suffers fundamental technical limitations [18]. First, it is unscalable. The rates at which device objects must be accessed and processed typically exceed the network/platform capacity. Platform processing and/or management communication resources can be quickly saturated as network size, speed, and complexity increase. Second, during stress times the platform must increase its interactions with agents, at a time when the network is least capable to handle these. Management response-time and reliability, furthermore, tend to stretch at a stress time, when fast and reliable response is most needed. Third, platform-centered management can lead to intense and unrealistic micromanagement of agents by platform applications. Fourth, platform heterogeneity and semantic heterogeneity, arising in the context of platform-centered management create barriers in the development of management applications.

Alternative management paradigms are needed to reflect the needs and opportunities of emerging networks and resolve the limitations of platform-centered management. Management should pursue flexible decentralization of responsibilities to devices and maximal automation of management functions through application software. Research toward such distributed management is described in Reference [18]. A management by delegation (MBD) paradigm is used to distribute management applications to device agents dynamically. Management application programs are delegated by platforms to device agents who execute them under remote platform control. MBD permits platforms to flexibly assign management responsibilities to devices, and even program devices to perform autonomous management. Additional research efforts to develop distributed management are described in References [13, 14].

Conclusions

It is useful to reconsider the central questions of network management: What should be monitored? How should it be interpreted? How should this analysis be used to control the network behavior? Management protocol standards provide syntactic structures to organize and access managed information. Such a syntactic framework can be useful in enabling systematic answers to these questions. However, the semantics of managed information, rather than its syntax, is the key to the answers. Clearly, the data that should be monitored needs to be derived from the model that is used to interpret the data. The model must be based on the semantics of the network operational behavior. Unfortunately, the manner in which interactions among network processes lead to faults or performance inefficiencies is not well understood. The storm example illustrated that, even when a fault behavior can be understood, it is unclear what symptoms need to be observed and how to correlate their respective data to handle it.

Significant research is needed to develop improved understanding of network operations and to build effective manageability. Standardization of managed information syntax is best viewed as a first step toward handling the semantics of network operations. Network management needs and scenarios are likely to continue and change as new types of networks, new applications, and better management technologies arise throughout the coming decade. As our understanding of the semantics of operations improves, new syntactic structures to support manageability will continue to emerge. Standardization of these mechanisms likely will continue and evolve in a manner not asimilar to the SNMP's evolution.

Acknowledgments

The author gratefully acknowledges the research support provided by the National Science Foundation (NSF) under project number NCR-91-06127.

References

[1] J. D. Case, *et al*., "A Simple Network Management Protocol (SNMP)," RFC 1157, May 1990.
[2] M. Rose and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets," RFC 1155, May 1990.
[3] K. McCloghrie and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II," RFC 1213, March 1991.
[4] J. D. Case, *et al*., "Introduction to the Simple Management Protocol (SMP) Framework," draft, July 1992.
[5] OSI, I.S.O., 10040 Systems Management Overview, 1991.
[6] OSI, I.S.O., 9595 Information Technology, Open Systems Interconnection, Common Management Information Services Definitions, 1991.
[7] OSI, I.S.O., 9596 Information Technology, Open Systems Interconnection, Common Management Information Protocol Specification, 1991.
[8] OSI, I.S.O., 10165-1 Information Technology, Open Systems Interconnection, Management Information Model, 1991.
[9] OSI, I.S.O., 10165-2 Information Technology, Open Systems Interconnection, Definition of Management Information, 1991.
[10] OSI, I.S.O., 10165-4 Information Technology, Open Systems Interconnection, Guidelines for the Definitions of Managed Objects, 1991.
[11] K. McCloghrie and M. Rose, "Common Management Information Services and Protocol over TCP/IP (CMOT)," RFC 1189, March 1991.
[12] M. T. Rose, The Open Book, A Practical Perspective on OSI (Prentice Hall, 1990).
[13] B. N. Meandzija and J. Westcott, ed., The First IFIP International Symposium on Integrated Network Management, (North Holland, May 1989).
[14] I. Krishnan and W. Zimmer, ed., The Second IFIP International Symposium on Integrated Network Management, (North Holland, April 1991).
[15] A. Kershenbaum, M. Malek, and M. Wall, eds., Network Management and Control Workshop, Tarrytown, N.Y. (Plenum Press, Sept. 1989).
[16] E. Horowitz, ed., Object-Oriented Databases and Applications (Prentice Hall, 1989).
[17] J. Ullman, Principles of Database & Knowledge Base Systems, vols. I & II, 3d ed. (Computer Science Press, 1988).
[18] Y. Yemini, G. Goldszmidt, and S. Yemini, "Network Management by Delegation," The Second International Symposium on Integrated Network Management, (North Holland, April 1991).

Biography

YECHIAM YEMINI has been a member of the Computer Science Department at Columbia University since 1980, where he presently serves as the director of the N.Y. State Center of Advanced Technology in computer and information systems. His main research interests include computer networks, network management, high-speed networks, protocols, distributed systems, and performance analysis. He has published and lectured extensively in these areas. Research at his Distributed Computing and Communications (DCC) lab resulted in network design and management technologies that have been widely exported, applied by hundreds of sites, and commercialized by leading industry. He has been a co-founder, director, and chief scientific advisor of Comverse Technology Inc., a successful public high-tech manufacturer of multi-media message communication computers. He is also a co-founder of System Management Arts, Inc., a recent New York start-up building network and system management software.