# Implementing OSI Network Management Facilities on the Xunet ATM Platform

*Nikos G. Aneroussis*
Center for Telecommunications Research
Columbia University
New York, NY 10027-6699
nikos@ctr.columbia.edu

*Charles R. Kalmanek* and *Van E. Kelly*
AT&T Bell Laboratories
Murray Hill, NJ 07974-0636
crk@research.att.com
vek@allegra.att.com

**Abstract**

We present the design of an OSI-based network management system for the Xunet ATM platform. We define a set of inheritance and containment trees and elaborate on the problem of linking logical objects in the Management Information Base with the corresponding network objects. Furthermore, we address several issues that are left open by the OSI management standards and require further consideration by the software developer, such as maintaining the consistency of logical objects, increasing the performance of the management agent, and recovering from failures. Finally, we identify the need for a more formal and precise model for network management that can take advantage of the many commonalities in the functionality of ATM networks to ensure portability between different platforms.

## 1. Introduction

Future ATM networks will require powerful management capabilities for configuration, performance and fault management. It is not only the large size but also the high level of complexity of the control model for such networks that calls for a new model of network management.

In this paper, we draw a distinction between network management and network control, based on their time scale of operation: Control operations typically operate with response times of the order of microseconds. Therefore, network control demands decentralized solutions that can affect the network in real-time [LAZ92a].

By contrast, the OSI model for network management usually involves the interaction of a human network manager through a set of management protocols, and therefore can operate at much slower time scales (of the order of seconds). Network management operations are based on measurements which can be used either for short term or long term actions. In the short term, these measurements are used to ensure proper network operations and guarantee the desired quality of service. In the long term, these measurements can be used for network planning and optimization. We regard OSI management systems as supporting *centralized monitoring and control* at slow time scales.

For example, management systems can monitor slowly changing network variables or obtain statistical information on fast changing variables such as time averages, variance etc. On the other hand, they are able to perform control operations that affect the macro-behavior of the network. The micro-behavior can be affected only by decentralized real-time control algorithms that do not use OSI management protocols. The network manager can modify the topology of the network by

Figure 1: The Xunet OSI management system.

enabling and disabling links, change virtual path routes, allocate bandwidth to virtual paths, etc. He can also alter the behavior of the scheduling algorithm by changing the parameters that define the scheduling policy. However, the manager cannot control in real-time the handling of individual cells. This can only be accomplished by a real-time scheduling algorithm.

By limiting management operations to affect only the macro-behavior of the network, it would seem that the size of the management problem becomes significantly limited. However, networks tend to grow larger, integrating heterogeneous hardware from different manufacturers with possibly different control interfaces. This fact increases the size of the modeling problem, since heterogeneity must now be taken into account. Furthermore, large networks demand a higher degree of availability. The manager needs tools that are simple to use but with sophisticated capabilities to monitor performance, detect and isolate faults, and restore the operation of the network as soon as possible.

We have started a network management experiment on the Xunet ATM platform. The purpose of the experiment was to study the implementation details of OSI management standards and discover issues that arise in the management of ATM networks. This paper is organized as follows: Section 2 describes briefly our network management implementation. Section 3 highlights areas that are not covered by the standards and in our opinion require further consideration in the design of management systems. Finally, Section 4 describes our future work.

## 2. Modeling the Xunet ATM Network

Xunet is a project between AT&T Bell Laboratories and several educational institutions that investigates the use of ATM technology and the concept of Quality of Service for multiple classes of traffic over a nationwide backbone of DS3 lines [FRA92]. Switches can be accessed through

200 Mbps host interfaces. A Silicon Graphics workstation serves as the switch Control Computer (CC). The CC runs the switch control software that performs signalling, control and fault detection functions.

Our objective was to design a small but expandable management system, offering a powerful set of network management capabilities. The management system consists of an agent part, which is installed in the CC of every switching node, and one (or more) manager(s) (Figure 1). The manager/agent interaction follows the OSI model [ISO88], and uses OSI protocols on top of TCP/IP connections. The functionality of the agent was limited to monitoring only as the first step, with control actions to be integrated later. The agent contains objects that monitor the activity of connections (virtual circuits) in every switch and the status of hardware sensors located throughout the switch. The agent is also capable of producing alarms through event forwarding discriminators, and sending them asynchronously to the manager.

The OSIMIS system [KNI91] was used as the programming infrastructure. OSIMIS provides source code for the implementation of a Generic Managed System (GMS) that closely follows the OSI standards [ISO91b, ISO91c]. A Management Service Access Point (MSAP) library is also provided as an interface to CMIS functions [ISO90a], used to implement the manager/agent interaction. Both the manager and the agent use these functions to exchange information.

OSIMIS was developed to monitor the OSI transport layer inside a Sun workstation and the original version had no capabilities to communicate with real resources outside the UNIX environment. We have expanded the system by adding new object classes for network objects, new attribute classes and new ways for communication between managed objects inside the Management Information Base (MIB) and their real counterparts. We have used the management tools that were provided with the OSIMIS distribution. An Xunet management tool is currently under development.

### 2.1 The Xunet Objects

The inheritance and containment trees for Xunet in this first implementation are shown in Figure 2. The `Network` Managed Object Class (MOC) contains the name of the network and other related attributes such as the network boot time. The `Node` MOC contains specific information about the node, such as the description of the switch fabric, the manufacturer, the location and the configuration of the local network, i.e. the Internet name of the switchboard, the type of the switch cabinet and the status of the controlling software. The `ConnectionManagement` MOC describes a generic object under which objects related to ATM connections are placed in the containment hierarchy. It maintains some information about the connection status of the switching node. Its presence is mostly justified from the need to categorize objects according to their functionality. Under this object we attach objects that represent Virtual Circuit (VC) connections, or the ATM adaptation layers. The `VcMgmt` MOC is used to describe an object that groups virtual circuit objects under a single parent. Furthermore, it maintains counters for the number of virtual circuits active for each traffic class. Only two classes are presently used, one for connections that require real-time guarantees and one for data. The `VcEntity` MOC is used to describe virtual circuits. It contains attributes for the incoming and outgoing slot and channel numbers, and other information related to protocol, line and switch states. In the future, we plan to add attributes that represent performance measurements for every VC.

In order to describe the status of the hardware boards that compose the Xunet switch, we have written the `HwEntity` and `XuDevice` MOCs. `HwEntity` is a very generic MOC that can be used to describe hardware modules in terms of manufacturer, manufacture date, hardware and software

Figure 2: The Inheritance and Containment Trees for Xunet.

version, etc. `XuDevice` is a MOC derived from HwEntity that describes hardware boards that plug into the Xunet switch. These boards are additionally characterized by attributes such as their position in the Xunet bus, the value of temperature sensors, etc. The board that contains the switch is described by the `SwitchBoard` MOC, which is derived from `XuDevice`. It has attributes that represent the errors that occur in the switch translation memory, errors associated with the ethernet interface, etc. These errors are further processed inside the agent. The agent will generate event reports if the appropriate Event Forwarding Discriminator objects have been previously created [ISO91i]. We have also planned the implementation of objects that describe other boards on the Xunet bus: Queue Modules, that do input and output buffering and cell-level scheduling, and Line Cards that perform the line interfacing task to the DS3 backbone links or the 200 Mbps host interfaces.

### 2.2 Linking with Real resources

In order to link the Xunet managed objects to their real network counterparts, new methods of communication with real objects were added to the OSIMIS system (Figure 3). The agent attaches to a shared memory directory, in order to interface to control processes running in the switch control computer. A new set of attribute classes were written so that the attributes of these managed objects can point directly through shared memory to the real resource that they represent. In this way, retrieving the value of an attribute always returns up to date information, since the attribute value is actually stored inside the real resource and is referenced through a memory pointer.

The shared memory interface is used in our experiment for connection-related objects (virtual circuits, etc.). Virtual circuits are maintained by a Call Processing process, located in the switch Control Computer. This process is attached to the memory address space of the OSI agent (running also in the same machine). The Call Processing process is responsible for setting up and tearing down virtual circuits and maintain the relevant information such as the source-destination pair,

Figure 3: Linking the MIB objects with Real resources.

etc. The agent can have always up-to-date information on what virtual circuits are active without having to explicitly communicate with another entity.

On the other hand, the information in the SwitchBoard object must be retrieved directly form the corresponding hardware unit using a UDP-based protocol over a local ethernet, that connects the control computer to the Switch board. The communication procedure is invoked every time the object is referenced, to obtain fresh information, and also automatically, over regular time intervals, so that errors in the hardware can be detected and the network manager informed. This polling function is internal to the agent and does not require activation from the manager side. An event report can be created asynchronously and sent to the manager when a fault occurs and some predefined conditions are met (an Event Forwarding Discriminator has been created).

## 3. Issues Arising from the Current Implementation

We have come across a number of implementation problems during the design of our network management system. The OSI standards describe in detail the protocol between the manager and the agent, the structure of the management information base and the operations on its contents. However, most problems arise from the fact that many implementation details have been left to the software developer. The central problem in our work is that the standards do not address at all the notion of time bounds in which management operations should be completed, both at the agent and at the manager side. As a consequence, the manager can experience very slow response times from the agent and receive "outdated" information. There is no way to enforce an acceptable level of performance in the management system. At the agent side, we have identified the following problem areas:

Figure 4: Maintaining Consistency between Logical and Real Objects.

## 3.1 Maintaining Consistency between Logical and Real Objects

The standards are mainly concerned with the logical representation of network objects in the MIB, and assume that operations on these objects are automatically reflected in the corresponding real network objects. In many cases, however, there is a delay while communication between the OSI agent and the real object is performed. As a result, the state of a real object is reflected in the corresponding logical object, and vice versa, with a certain time delay.

In Xunet, the agent is running on a workstation, and is accessing the switch hardware through an ethernet-based interface. Logical objects are implemented inside the workstation's memory, and must reflect the state of the corresponding real network objects. Maintaining consistency in our system is very important because event reports are conditionally generated when the state of a real object changes and without prior interaction with the manager. If the logical object is not updated frequently, event reports will be generated with a significant delay or not generated at all. We are more concerned about how to reflect the state of the real object in the logical object, since the reverse is usually quite simple. Consistency can be achieved using any of the following methods of communication (Figure 4):

1. The agent polls the real object at regular time intervals. This implies that communication is established with the real object, and the acquired information is used to update the MIB. At this point, alarms may be triggered, if a preset fault condition is satisfied. A variant of this approach is to poll the real object only when the manager performs a `get` operation on the logical object. In both cases, it is possible that the agent misses an event, if the interval between successive polls is large.

2. The real object asynchronously calls the agent when a change of state occurs using, e.g.,

a Unix signal. The agent then establishes communication with the object and updates the necessary information. This is the fastest way to update the state of the logical object. The main drawback of this approach is that the agent can be overburdened by a large amount of redundant state updates.

3. The real object sends a message to the agent that contains the new state of the object. The agent however will not necessarily receive this message immediately, but at his convenience, when he checks for input. By that time, it is possible that the information may be already outdated, especially if the variables involved change in a faster time scale.

4. The real object attributes are linked through a shared memory interface directly into the logical object inside the MIB. This is only possible if the real object and the OSI agent reside in the same workstation or distributed environment that supports memory sharing. When the shared memory interface is used, a `get` operation on the object returns immediately up-to-date information. However, it is not straightforward to handle asynchronous events. For example, when an attribute value changes, the agent has no way of knowing this, unless he is notified by a signal or polls the object in regular time intervals, as mentioned above. Polling in this case does not involve a communication delay, and therefore can reduce overhead.

The above methods of communication work well for static objects, i.e., objects that exist throughout the lifetime of the agent. Objects that are created and destroyed dynamically need special handling. Now, in addition to updating the state of the logical object, we must notify the agent if an object has been deleted, or a new object has been created. In our system, virtual circuit objects are created and destroyed dynamically. We can use two approaches: According to the first approach, the agent must check periodically to see if such objects have been created or deleted. The second approach uses an ad-hoc protocol to asynchronously notify the agent of object creation and deletion events.

## 3.2 Improving Agent Efficiency through Multithreading

The system that we have used can support many open connections to OSI managers, but serve only one of them at a time. Furthermore, communication with real objects has blocking semantics, i.e., the agent blocks until it receives a reply from the real object. (This does not hold for the shared memory interface.)

It is true that with the information obtained from real objects, sometimes, more than one object can be updated. However, an agent for an ATM network may have a very large number of objects that represent heterogeneous resources, and have to communicate with each of them frequently. This implies that the agent could be blocked for a large percentage of time.

Our intention is to provide a *multithreaded* agent that will resolve to a substantial extent this problem. By providing multiple threads of control, communications can be dynamically delegated to threads, and the agent can continue internal processing tasks (such as event reporting and real object polling), or servicing requests from OSI managers (Figure 5). However, a multithreaded implementation needs additional considerations:

- Since multiple threads must use the kernel communication facilities at a time, more than one thread must be allowed to make system calls at a time. However, this feature is available in very few operating systems that provide kernel-level support for threads. As an alternative, we could use a non-blocking I/O library within a single-threaded environment.

Figure 5: Parallelism increases Agent Performance.

- Maintaining a certain level of consistency within the MIB is now more difficult. For example, in a multithreaded system, many managers are allowed in theory to alter attributes of an object at a time. Also, a `set` operation on an object may be overridden by a previous `get`, that has already initiated communication with the real resource, and is now waiting for a reply to update the object. This is also known as the "lost update" problem, which also occurs in database systems. Furthermore, even if it were possible to synchronize concurrent access to individual objects, it would be difficult to guarantee the consistency of management operations that involve more than one managed objects, if such operations intersect. The solution to these problems requires support for distributed transactions. Each transaction models the interaction of the manager with possibly more than one agent, and guarantees the consistency of concurrent management operations. However, this is a difficult problem, because CMIP is not a transaction-oriented protocol and does not explicitly support transaction-based concurrency control. It appears to be much easier to try to enforce coordination between management applications than to extend the OSI management model to support concurrency control.

### 3.3 Managed Object Persistence and Recovery

The OSIMIS system implements the Management Information Base in a workstation's volatile memory. It is possible for the agent to crash, without anything happen to the network. In this case, it is desirable to recover the agent to its previous state before the crash. The MIB has to be reconstructed in its entirety. Both the objects that existed before the crash must be recreated, and their attributes obtain their previous values. An agent crash may lead to the following inconsistencies:

Figure 6: Introducing Managed Object Persistence.

- Set operations from the manager, may have been propagated to the network, but the manager might think that they have failed since no notification was received. In this case, the manager must either be notified, when the agent is revived, or the operations must be undone.
- Objects such as Event Forwarding Discriminators that have no real resource counterpart and therefore cannot be recreated from information received from real objects, must be restored to their previous state.

A persistent object is an always consistent copy of the logical object which resides in permanent storage (e.g. a disk). When managed objects have this property, it is possible to solve effectively the above inconsistency problems (Figure 6):

In the first case, the logical object can be reconstructed by polling the real object or by recovering from the persistent object (which can also record the fact that the manager is awaiting for a notification). In the second case, the discriminator object is reconstructed from the persistent object, and as a result, the manager will continue to receive event reports.

Another solution to the recovery problem is to delegate the problem of reconstructing the agent state entirely to the manager. The manager is always notified when his association to the agent has been broken. When the association has been reestablished, the manager can restore the state that cannot be recovered, such as event forwarding discriminator objects etc. However, this solution may run into problems, when more than one managers are connected to the network, and each one attempts to recover the agent to an acceptable state.

Another advantage of persistent objects, is that their state trajectory can be examined at any time. It is possible from the manager side to retrieve images of the managed object at different times, and from them extract a representation of the object state trajectory. Some of this functionality

Figure 7: The model for developing a management system.

can also be incorporated inside the agent, using the workload monitoring function. This function allows us to extract statistical information such as time averages etc. from object attributes, without requiring object persistence.

## 4. Future Work

The system that we have built has served as a valuable experimental platform for understanding OSI network management concepts. During the time this work was done, the structure of the control software for the Xunet network changed, and we are currently in the middle of revising our network management software.

We have implemented as managed objects only a small factor of the entities that need to be monitored and controlled inside an ATM network. It has turned out to be hard to come up with a good model that describes an ATM network through a set of inheritance and containment trees. The process of transforming a fuzzy, prototypical view of concepts into a precise and formal object representation is informal and cannot be formalized. Therefore, there are only intuitive but not formal models that can distinguish a better object representation of the network from a worse one.

Therefore, our modeling problem with respect to the design of the inheritance and containment trees should be guided by our objectives for monitoring and control. An integrated model for management should consider the following (Figure 7):

- The design of the Management Information Base: For each desired management function that falls into one of the five OSI functional areas (configuration, performance, accounting, security, fault management), we must identify logical network objects, define their attribute structure and behavior, and integrate them into the management system. One logical object may support more than one management function, and therefore we must avoid the definition of redundant objects.

- The definition of the required level of consistency between the real object and its logical representation. This includes the behavioral definition for the managed object regarding the communication with the real object. The system designer must take into account the communication latency between these two entities.

- The definition of a monitoring model for the network. The monitoring model identifies what parameters of the system need to be monitored and if statistical processing needs to be applied. Furthermore, the monitoring model describes what network events are of monitoring interest to the manager. This function can be implemented through Event Forwarding Discriminator constructs that filter the primary monitored information and produce alarms that are transmitted asynchronously to the manager.

- The definition of a control model for the network. The control model identifies the control operations that can be carried on the network and is reflected on the management system by the ability to set object attributes and perform actions on logical objects that will in turn be propagated to the corresponding real objects.

We will continue our work along these lines to come up with an integrated model of management for the Xunet network. Since ATM network architectures differ substantially from each other, it is not easy to agree on a common set of inheritance and containment trees, and on a common monitoring and control model. Furthermore, differences between implementations require different managed object behavior. It is very difficult to develop a compiler that can take as input a high level description of an object's behavior and produce the implementation code, especially the code which communicates with the real resource. For this reason, behavioral information is usually hard coded into the management agent. As a result, source-level portability of management agents between different platforms is difficult. However, there are many commonalities between management models for ATM networks that stem from their similar functionality, and this may lead in the future to a commonly accepted (but probably higher-level) model for network management.

## 5. Conclusions

We have started the implementation of OSI management standards within a network management system for the Xunet ATM network. The most important aspect of the management problem is the definition of a clear model for network management, that identifies managed objects and their attributes, the behavior of these objects, and a set of management functions (or algorithms) that implement the management policy (the monitoring and control model). In the course of our work in this area, we have discovered several issues, regarding data consistency between the entities that compose the management system, and its performance characteristics. The solution to these problems will require either further standardization or, more likely, commonly accepted solutions.

## References

[BLA92] Uyless Black, *Network Management Standards: the OSI, SNMP and CMOL Protocols.* New York, NY: McGraw Hill, 1992.

[DIN92] John Dinger and Derick Jordaan, "Experiences with the use of OSI Management Infrastructures," in *Proceedings of the 1992 Globecom*, December 1992.

[ELM89] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems.* Redwood City, CA: Benjamin/Cummings, 1989.

[FRA92] A.G. Fraser, C.R. Kalmanek, A.E. Kaplan, W.T. Marshall and R.C. Restrick, "Xunet 2: A Nationwide Testbed in High-Speed Networking," in *Proceedings of the IEEE Globecom*, Florence, Italy, May 1992.

[GOS91] Andrzej Goscinski, *Distributed Operating Systems: The Logical Design.* Addison-Wesley, 1991.

[ISO88] Information Processing Systems - Open Systems Interconnection, "OSI Management Framework," October 1988. International Standard 7498-4.

[ISO90a] Information Processing Systems - Open Systems Interconnection, "Common Management Information Service Element (CMISE)," January 1990. International Standard 9595.

[ISO91b] Information Processing Systems - Open Systems Interconnection, "Structure of Management Information - Part 1: Management Information Model," July 1991. International Standard 10165-1.

[ISO91c] Information Processing Systems - Open Systems Interconnection, "Structure of Management Information - Part 2: Definition of Management Information," August 1991. International Standard 10165-2.

[ISO91i] Information Processing Systems - Open Systems Interconnection, "Systems Management - Fault Management - Part 5: Event Report Management Function," July 1991. International Standard 10164-5.

[KNI91] George Pavlou Graham Knight and Simon Walton, "Experience of Implementing OSI Management Facilities," in *Integrated Network Management, II* (I. Krishnan and W. Zimmer, editors), pp. 259–270, North Holland, 1991.

[LAZ92a] Aurel A. Lazar, "The Integration of Real-Time Control with Management in Broadband Networks," in *Proceedings of the Workshop on Broadband Communications*, Estoril, Portugal, January 1992.

[TSU92] Mitsuru Tsuchida, Aurel A. Lazar, and Nikos G. Aneroussis, "Structural Representation of Management and Control Information in Broadband Networks," in *Proceedings of the IEEE International Conference on Communications*, Chicago, IL, June 1992.