Integrated Network Management, II
I. Krishnan & W. Zimmer (Editors)
Elsevier Science Publishers B.V. (North-Holland)
© IFIP, 1991

Experience of Implementing OSI Management Facilities

Graham Knight , George Pavlou and Simon Walton

Department of Computer Science, University College London, Gower Street London, WC1E 6BT, United Kingdom

ABSTRACT

The Computer Science department at UCL has experimented with OSI management systems for several years and has implemented a pilot management system on a Unix workstation. A second version of this system is now being implemented. This paper briefly reviews our experience with the pilot system and outlines the capabilities that were felt desirable in its successor. The architecture of the successor system is then described.

1. INTRODUCTION

The Computer Science department at UCL has experimented with OSI management systems for several years and has built a pilot system. We have been interested in investigating how practical is the OSI approach when compared with that of (say) SNMP[1]. In particular, we have wanted to experiment with the complex filtering and event control facilities that OSI management provides.

Section 2 describes the key features of the pilot system and Section 3 outlines the areas in which development was felt to be necessary. Section 4 gives an overview of the architecture of the updated system whilst Sections 5-7 provide a detailed description of its internal operation. It is assumed that the reader is familiar with the basic concepts of OSI management. A good tutorial introduction to these may be found in [2].

2. THE PILOT SYSTEM

Our first implementation "Osimis" [3] was developed under the ESPRIT INCA project[4] and was designed to provide OSI Management facilities for a Unix workstation. It included a CMIP implementation to the DP of 1988 built upon ISODE[5]. This system is illustrated in Figure 1; it provided communication between a Unix "Systems Management Agent" (SMA) and three clients; an event logger ("Osilog"), a status monitor ("Osimon") and an MIB browser ("Osimic"). The MIB was restricted solely to the ISODE Transport Layer.

Some of the key features of this system are described below.

2.1. Managed System Internal Communication

The ISODE Transport protocol code that was used ran in user space. At any given moment, there could be several instances of the protocol active within several Unix processes. In order to de-couple management protocol operation from communications protocol operation and to collect management services in one place, it was decided to implement the management services in a single process - the "Systems Management Agent" (SMA).

The problem then arose of how and when management information should pass between the communications protocol processes and the SMA. In the event, a UDP socket was chosen, with all communication being initiated by the Transport protocol processes. The Transport protocol code was liberally seeded with entry points to the management code that would trigger the dumping of management information to the SMA at significant moments. Triggering events included T-Connects, T-Disconnects, a fixed data quantity transferred etc. This avoided the problem of having to interrupt Transport protocol operation in order to deal with asynchronous requests from the SMA - a procedure that was held likely to have unpredictable effects on Transport (and other) protocol operation. Unfortunately, this arrangement also made it difficult to pass information from the SMA to the protocol processes. Since our main interest at the time was in event-driven management we decided that one-way communication was acceptable.

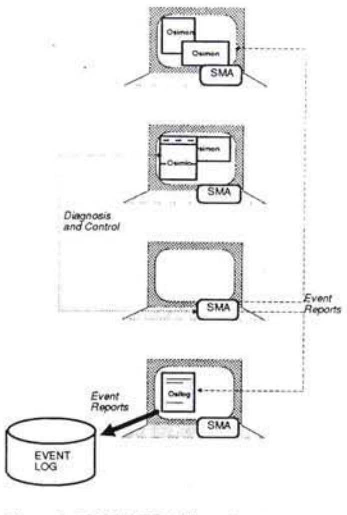

Figure 1. UCL "OSIMIS" version 1.

2.2. Event Report Control

At the time of the original implementation, the management of event reporting was not very complete in the standards. There was a notion of a "Defined Event" - an attribute of a Managed Object (MO) which corresponded to some event in the real world, a Transport Disconnect for example or a management event such as a threshold being exceeded. There was also a "Report Control" MO which was tied to a Defined Event and specified the information to go in the report, together with a list of recipients.

Having Report Control and Thresholds as MOs in their own right meant that, in principle, these could be created and deleted dynamically and that many-to-many relationships could exist between Defined Events, and Threshold and Report Control MOs. This suited the way in which we proposed to use event reports which were seen as a general purpose tool for driving loggers, status displays and diagnostic tools.

Although the main purpose of the system was to support human "managers", it also needed to cater for the requirements of our relatively sophisticated users who like to know what is happening on the department's systems. If performance seems poor, they like to be able to find out why by getting an up-to-date picture of activity. The "netstat" program on Berkeley Unix systems is an example of one which addresses this need. We wished to satisfy a similar requirement for ISODE. If facilities like these are to be event-driven, then it must be possible for several

remote systems to receive event reports from a target system simultaneously and for the event reporting criteria (thresholds etc.) to be individually tailored. This requirement is in contrast to the more conventional one in which is only necessary to report events to a single remote sink.

2.3. The MIB Browser

The aim of the browser was to provide a tool that would be useful to system developers and maintainers who needed to focus in on some aspect of the operation of an OSI component and view its operation in detail. In OSI Management terms this means providing a detailed view of a single MO. The problem we faced was in the naming of transient MOs such as those representing Transport Connections. These are named by identifiers (usually integers) allocated by the parent system at "create" time. Their scope is purely local and there is no way that an external management process can know of these *a priori*. The browser provided a graphical interface to the tree of MOs on a remote system. From any object, it was possible to list the MO classes and Relative Distinguished Names of the subordinates. In the case of transient objects, the subordinates could be examined in turn until the required one was found.

3. REQUIREMENTS FOR THE SUCCESSOR SYSTEM

When the time came to update the pilot system we had several things in mind:

i) The standards had developed further, this was particularly noticeable in the area of event control, "discriminators" having been introduced in order to enhance this. These had some of the properties of our Report Control MOs but with much more sophisticated filtering.

ii) Our original CMIS/P implementation was not quite a full one. The M-CREATE, M-DELETE and M-ACTION services were omitted as was filtering, scoping and wildcarding. CMIS and CMIP had now stabilised at the final DIS stage and differed substantially from the versions we had used. An update was essential.

iii) It was clear, even from our small-scale pilot scheme, that OSI management systems were inclined to be large - too large for many small but high-performance network components. We were keen to investigate the use of proxy systems in these circumstances.

iv) Though it was clear that the MIB we managed would need to be expanded to include additional components, we could not say in advance precisely what these should be.

v) Our own MO definitions were "interim" at best, skeletal at worst! We needed to be able to adapt our MO definitions as the standards process proceeded, with a minimal disruption to existing code.

vi) The proprietary management facilities inherent in network components are rarely designed with OSI management in mind - nevertheless we wanted to incorporate these. Quite complex data mappings are sometimes needed in order to massage native management information into an OSI-like form. Further, proprietary management protocols vary in the message set they offer and in fundamentals such as which party initiates communication. We needed to construct systems which were flexible enough to allow new components to be incorporated no matter what proprietary facilities were on offer.

vii) Notwithstanding the variety of components alluded to above, many features of OSI management are common to all components. We wished to extract the common facilities into a few tightly-defined modules that could be used as a basis for building a variety of systems by a variety of people.

4. GENERIC MANAGED SYSTEM OVERVIEW

To date, in our design of a successor system, we have concentrated on the Managed System and how this can be structured with extensibility in mind. We have attempted to provide the generic features of an OSI Managed System together with a support framework which may be used by the implementors of MOs. The target environment is a Unix workstation requiring OSI management facilities and which may also act as a proxy for another machine. The result we call the "Generic Managed System" (GMS).

The internal structure of GMS is shown in Figure 2. To a large extent, the structure reflects the OSI model of a Managed System so that the major software interfaces correspond to those of the OSI model. For example, the external CMIS interface specified in [6] and the internal "object boundary" interface outlined in [7] are both represented by software interfaces. However,

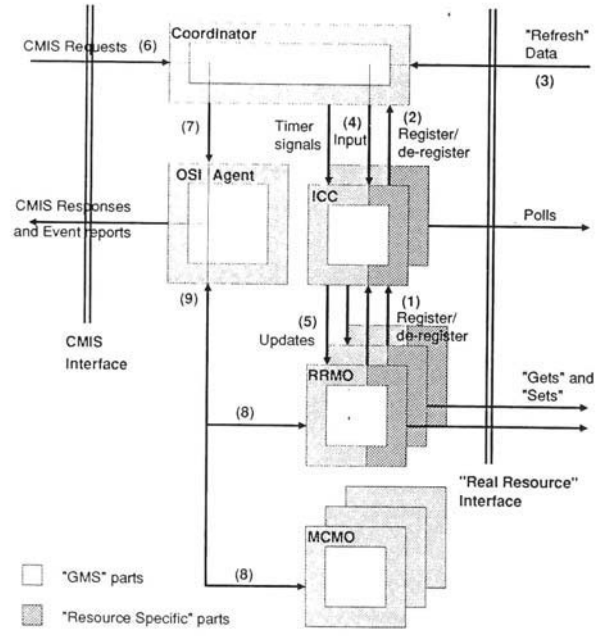

Figure 2. The UCL Generic Managed System

the OSI model was never intended to be an implementation model and significant divergencies have been made in order to arrive at a practical design.

The GMS is implemented as a single Unix process; the implementation language being C++. There are five major software components, each realised as a C++ object or set of objects:

i) Real Resource Managed Objects
ii) Management Control Managed Objects
iii) Internal Communication Control Objects
iv) The Coordinator.
v) The OSI Management Agent

These are described in the sections below.

5. INTERNAL STRUCTURE AND FUNCTIONALITY

5.1. Managed Objects

From the implementation point of view, two sorts of MO classes may be identified. The first are abstractions of "real resources" (Transport connections for example) which need to be managed; these we call Real Resource MOs (RRMO). The second relate to features of the management system itself and exist so as to allow the operation of the management system to be controlled via standard management operations; we call these Management Control MOs (MCMO). One example of a MCMO is an Event Report Forwarding Discriminator (ERFD). This contains information specifying which events should be reported and to where. Event reporting behaviour can be modified by changing this information through the use of CMIS M-SET operations.

5.1.1. Real Resource MOs

Implementations of "Real Resource" MO classes (RRMO) may be considered to have two parts (see Figure 2):

i) A part common to all RRMO classes. This is provided by the GMS and includes:

- A C++ object class for a generic MO. This has methods corresponding to the MO boundary interface plus some additional ones to assist with maintenance. Specialised subclasses of this may be derived as required.
- C++ object classes for commonly occurring attributes such as counters, gauges, thresholds and tidemarks. In fact, all the attribute types in [8] are supported in this way.
- A support environment to assist with and coordinate communication with the real resource. This environment is described more fully in Section 5.2

ii) A part specific to a particular RRMO. This must be tailored not only to the real resource type but also to the means the real resource uses to present management information. These "resource specific" parts are not provided by the GMS and must be supplied by the individual implementors of RRMO classes.

5.1.2. Management Control MOs

MCMO classes are common to all management systems no matter what real resources are being managed. Hence, GMS provides implementations of MCMO classes in their entirety.

At present, the only MCMO class provided is the ERFD one. ERFD objects may be created, destroyed and updated as a result of CMIS messages from a remote manager.

5.2. Communication with Real Resources

We now consider the ways in which management information may be obtained from real resources. Real resources may reside in the operating system's kernel, on communications boards, in user-space processes or even at remote systems which are managed via proxy management. The information they contain may be accessed by reading the kernel's virtual memory, talking to a device driver, communicating with another user-space process using an IPC mechanism or - in the case of proxy management - with a remote system using a communications protocol. In general, communication needs to be two-way as it should be possible to perform "intrusive" management by setting management information values in the real resources.

From the point of view of the GMS, information flow may be triggered:

i) asynchronously as a result of some activity on the real resource.

ii) by a timeout indicating that a real resource should be polled.

iii) as a result of a CMIS request from a remote manager process.

Each of these embodies a trade-off between the timeliness of the management information that the GMS can make available and its responsiveness. Used exclusively, iii) makes event reporting infeasible and implies the operation of a pure polling regime by the manager such as is favoured for SNMP.

5.2.1. Internal Communications Control

If generality is to be achieved, the GMS must support all the communications methods above, maintaining at the same time well-defined and uniform interfaces between the RRMOs and the rest of the system. It must be remembered that several RRMOs may be associated with a single real resource; ideally such a "family" of RRMOs should share a single communications path to the real resource. In order to achieve this, the notion of an Internal Communications Control (ICC) object is introduced. An ICC object coordinates the updates of a family of RRMOs that are realised in a similar fashion. ICC objects are repositories for information about the mode of communication to be employed, they initialise this communication and understand conventions such as the nature and structure of the messages exchanged, i.e. the protocol used. ICCs are created at system start-up time for each RRMO family that is to be managed.

As there are no real resources associated with MCMOs these do not have corresponding ICC objects.

5.2.2. The Coordinator

Given that several real resources are being managed and that messages are also being sent and received across the CMIS interface, it can be seen that some organisation is necessary to ensure that incoming messages are delivered to the correct objects and that no object can do a blocking read thus disabling the whole system. This is achieved by ensuring that all incoming messages are delivered first to a "Coordinator" object which then distributes them.

When the first RRMO in a family is created (either as a result of a CMIS M-CREATE request or of some activity on the real resource), its ICC interacts with the Coordinator in order to register an endpoint of communication (typically a Berkeley socket) to the real resource through which asynchronous messages may be expected.

An ICC may ask the Coordinator to call one of its methods at regular intervals so that it may poll the real resource. Alternatively, it may ask that whenever data becomes available at the communication endpoint a method should be called. Typically, this method will read the incoming data and pass this to the correct RRMO. The only case when RRMOs interact directly with the real resources is when they set management information.

5.3. The OSI Agent

The other major component of the OSI managed system model is the "OSI Agent". This too is represented by a C++ object and handles wild-card naming, scoping, filtering and (eventually) access control.

The OSI Agent services the messages it is handed by the Coordinator. These may be either association establishment/release requests or CMIS operation requests. In the latter case it first performs access control functions and then synchronises the potentially multiple replies according to the scoping and filtering parameters. In order to perform CMIS requests, it interacts with the selected MOs to get, set, etc. management information.

The OSI Agent may also receive event "notifications" from the RRMOs. According to the OSI management model, MOs issue notifications to the agent which then checks with the event filtering information in the ERFDs to determine whether the notification should result in a CMIS **M-EVENT-REPORT.** Unfortunately, if an implementation follows this model it results in a great deal of wasted processing in the case that no remote manager is interested in the event in question. There are also some logical problems; for example, the filtering expression may reference the MO that issued the notification but this may, by now, have been destroyed. Within the GMS, ERFD filtering information is applied in advance and notifications are only issued by RRMOs if it is known that **M-EVENT-REPORT**s will result. Although we have implemented the full generality of the filtering mechanism specified in the draft standards[9], we can see that certain filters will be extremely expensive to process. We expect that, in practice, only quite simple filtering expressions will be used.

6. METHODS

As an aid to understanding the information flow within the GMS we now summarise the methods applicable to the objects above. The C++ `obj.method` notation is used (somewhat loosely) to indicate a method `method` being applied to an object with id `obj`.

6.1. The OSI Agent

Three methods are used by the Coordinator to report incoming messages from the CMIS interface:

```
assoc_id = agent.cmis_connect (connect_ parameters)
agent.cmis_work (assoc_id)
agent.cmis_lose (assoc_id)
```

The first is used to notify a request for the establishment of a CMIS association, the second to indicate that a message has arrived on an existing association, (including a disconnect request), and the third to indicate that an existing association has been abnormally released.

A further method is used by the RRMOs to notify events:

```
agent.notify (my_class, my_name, event_type, event_report_info,
                              destination_address_list)
```

6.2. Managed Objects

The OSI agent interacts with the RRMOs and MCMOs to perform requested CMIS operations. The procedures and methods used are:

```
result = Create (parent_MO, rdn, init_info)

result = mo.Get (attribute_ids)
result = mo.Set (attribute_id/value pairs)
result = mo.Action (action_type, action information)
result = mo.Delete ()
```

Create is a static method which checks whether a create request is valid and, if it is, calls the constructor for the appropriate C++ class. The identity of the parent MO in the containment hierarchy and the Relative Distinguished Name (RDN) of the new MO are supplied as parameters. The four methods shown then embody the interface defined in [7]. The

Four other methods are provided to assist the OSI agent in locating the required MO:

```
target_mo = mo.find (name)
mo_list = mo.scope (scope_info)
answer = mo.filter (filter)
answer = mo.check_class(my_class)
```

The first searches the subtree below **mo** for a MO called **name**, the second returns a list of MOs which are "in scope" according to **scope_info**, the third applies a filter to a MO and returns a boolean value, the fourth checks that the class **my_class** is appropriate for the MO in question.

An ICC may need to create or delete transient RRMOs and to refresh the RRMOs it controls with new management information according to activity in the real resources. RRMOs are created by the ICC calling the constructor directly. The methods used by ICCs are:

```
mo.do_update (management information)
mo.destructor ()
```

6.3. The Coordinator

The ICCs tell the coordinator to register or de-register endpoints of communication to the real resource and are subsequently informed of activity on these. They also tell the coordinator to schedule and cancel periodic polling signals. The methods used are:

```
coord.register_cep (icc, cep_id)
coord.deregister_cep (icc, cep_id)

coord.schedule_poll (icc, interval, MOclass)
coord.cancel_poll (icc, MOclass)
```

6.4. ICCs

RRMOs in a family register and de-register themselves with their ICC as they are created and deleted:

```
icc.register_object (mo, mo_class, rdn)
icc.deregister_object (mo)
```

Note that the first RRMO to register triggers the establishment of communication to the real resource.

RRMOs may optionally request a special polling regime for a particular MO class and these requests are passed to the Coordinator via the ICC:

```
icc.schedule_poll (interval, MOclass)
icc.cancel_poll (MOclass)
```

A RRMO may need to talk directly to the relevant real resource - for example when the setting of some attribute value should rapidly be reflected in system operation. In this case, the RRMO must ask for its communication end-point from the ICC:

```
cep_id = icc.get_cep ()
```

The Coordinator needs to inform ICCs of the arrival of a message from a real resource or of the necessity to issue a poll. These two methods are used:

```
icc.do_cepread (cep_id)
icc.do_poll (MOclass)
```

The object class parameter in the poll method is only used when the polling takes place for a single MO class within a family rather than for the family as a whole.

7. THE GMS IN USE - AN EXAMPLE

The first RRMO to be implemented was a port of the ISODE TP0 management functions from the old OSIMIS system. An important test of the GMS structure was the ease with which this could be done.

In the case of ISO TP0, we identified two MO classes: the T-Entity class and the T-Connection class. There may be one and only one (static) instance of the T-Entity class. This summarises activity for all incarnations of the protocol and contains information such as the number of current and previous connections, the amount of data transferred, and error counters. Instances of the T-Connection class are transient - existing only during the lifetime of a connection. They contain information such as creation time, source and destination TSAP addresses and traffic counters. These are subordinate to the T-Entity instance in the MIB containment hierarchy.

ISODE TP0 is implemented as a set of library routines that are linked with the applications, this means that it runs in user space - the "real resource" in this case is effectively a Unix process. The IPC method used to communicate management information is a UDP socket - communication is only possible from the real resource to the GMS at present.

Implementation was straightforward. C++ object classes for the T-Entity and T-Connection RRMO classes were derived from the generic C++ MO class. Many of the additional attribute types required were instances of C++ classes already available within the GMS. An ICC object class was written (again, derived from the generic C++ one). This registers a socket bound to a well-known port with the Coordinator. An ICC method (`icc.do_cepread()`) is then called by the Coordinator each time a message arrives at the socket. Detailed operation is as follows (the numbers in the text are references to Figure 2).

When the T-Entity RRMO is created, (which happens either at initialisation time or through a CMIS M-CREATE request), it registers itself (`icc.register_object ()`) with the "ISODE" ICC object (**1**). If it is the first ISODE-related RRMO to register, the ISODE ICC object initialises the UDP socket, so that it may be contacted by active ISODE processes. It also registers this "communication endpoint" with the Coordinator (`coord.register_cep ()`) so that it will be notified in case of activity (**2**). After this, T-Connection RRMOs may be created and these too will be registered with the ISODE ICC.

When a message arrives at the UDP socket from an ISODE process (**3**), this is fielded by the Coordinator which recognises the socket as being managed by the ISODE ICC which it then informs (`icc.do_cepread ()`) (**4**). The ISODE ICC then passes the incoming information to the relevant RRMOs (**5**) (`mo.do_update ()`).

A CMIS M-GET request will also be fielded by the Coordinator (**6**) and, in this case, will be passed to the OSI Agent (**7**) (`agent.cmis_work ()`) which will perform the scoping and filtering tasks in order to select the relevant MOs. It then performs the requested operation (**8**) (`mo.xxx()`) on these.

As a result of processing information received from an ISODE process, an RRMO may determine that a threshold has been exceeded and that a notification may be required. If the RRMO determines that an ERFD filter is set to forward such a notification, it informs the OSI Agent (**9**) (`agent.notify ()`).

Finally, when an ISODE RRMO is deleted, (which happens, for example, when a T-Connection closes or as a result of a CMIS M-DELETE request, it deregisters itself with the ISODE ICC object. If it was the last RRMO, the UDP socket is closed and the Coordinator is notified accordingly, so that future protocol instances will not talk to the agent.

8. CURRENT STATUS AND FUTURE PLANS

We have, at present, an implementation of the GMS as described. The only RRMOs supported so far are those related to the ISODE implementation of ISO TP0 described above.

The next step will be to add RRMOs related to further classes of real resource. One of the first of these will be the Berkeley Unix TCP/IP implementation. Although it might seem odd to manage a non-OSI protocol suite in this way, its extensive use in our environment means that it will exercise the GMS in a realistic way.

The OSI management work is being undertaken as part of the ESPRIT project "PROOF"[1]. This project is building a connection oriented Ethernet ISDN gateway, UCL is also building a connectionless version. Both of these gateways will be managed by using the GMS as a proxy; ISDN, X.25 and Ethernet RRMOs will be needed.

Another possibility we will look at is the implementation of RRMOs with SNMP back-ends. These would enable the GMS to operate as a proxy system for SNMP agents, enabling these to be managed by OSI manager processes.

The standards continue to develop, one [8] has been superceded [10] and no doubt others will suffer the same fate. We intend to modify the GMS as soon as these changes seem to be fairly stable.

Finally, we will investigate the use of a managed system specification language with a compiler to generate code for the generic parts of RRMOs, to define a MO schema and to provide initialisation data for static MOs. In this way, only the "back-end" code for interacting with the real resources will need to be hand written.

We do not claim that this system provides all the answers for OSI management. The GMS was built as an experimental tool with flexibility rather than performance and compactness in mind. However, we do hope that it will give us some practical insights into the problems of OSI management which will be valuable in the future, and that our experience will be useful to others facing similar problems.

1 The PROOF partners are: 3Net (UK - Prime Contractor), SNI (Germany), System Wizards (Italy), University College London (UK).

REFERENCES

1 Case, J.D., Fedor, M., Schoffstall; M.L., Davin, C. "Simple Network Management Protocol (SNMP)", (DARPA RFC 1098), April 1989

2 Sluman C., "A tutorial on OSI Management", "Computer Networks and ISDN Systems" vol 17 pp 270-278, 1989

3 Knight G. J., "The INCA Network Management System", Connexions - The Interoperability Report, Vol. 3, no. 3, pp. 27-32, March 1989

4 Knight G.J., Kirstein P.T., "Project INCA - An OSI Approach to Office Communications", Proceedings of the OSI87 Conference, pp 245-254, Online, London 1987.

5 Marshall T. Rose, "The ISO Development Environment User's Manual", U. Delaware, 1990

6 ISO/DIS 9595 (Final Text N3874), "Information Technology - Open Systems Interconnection - Common Management Information Service Definition", 2nd DP N 3070, January 1990

7 ISO/DP 10165-1 (Proposed DP Text), "Information Processing Systems - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 1: Management Information Model", June 1990

8 ISO/DP 10165-3 (Proposed DP Text N 3302), Information Processing Systems - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 3: Definitions of Management Attributes, January 1989

9 ISO/DP 10164-5 (N3845), Information Processing Systems - Open Systems Interconnection - Systems Management - Part 5: Event Management Function, September 1989

10 ISO/DIS 10165-2, Information Processing Systems - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 2: Definition of Management Information, June 1990