.

THE OSIMIS TMN PLATFORM: SUPPORT FOR MULTIPLE-TECHNOLOGY INTEGRATED MANAGEMENT SYSTEMS

George Pavlou

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

Abstract

The *OSIMIS (OSI Management Information Service)* platform provides the foundation for the quick and efficient construction of complex multiple technology management systems. It is an object-oriented development environment in C++ [1] based on the OSI Management Model [2] which hides the underlying protocol complexity (CMIS/P [3][4]) and harnesses the power and expressiveness of the associated information model [5]. OSIMIS combines the thoroughness of the OSI models and protocols with advanced distributed systems concepts pioneered by ODP to provide a highly dynamic distributed information store. It also combines seamlessly the OSI management power with the large installed base of Internet SNMP [6] capable network elements.

OSIMIS is ideally suited for Telecommunication Management Network (TMN) [7] environments because of its support for hierarchically organised complex management systems and its ability to embrace a number of diverse management technologies through proxy systems. OSIMIS provides a generic CMIS to SNMP application gateway [8][9] while adaptation to other models or proprietary systems is feasible through its proxy system support. OSIMIS projects a model whereby OSI management, being the most powerful of available technologies, provides the unifying end-to-end means through which other technologies are integrated via application gateways, possibly in a generic fashion. This paper explains the OSIMIS components, architecture, philosophy and direction.

Keywords: Network, Systems and Application management; Distributed Systems

1. Introduction and Overview

OSIMIS is an object-oriented management platform based on the OSI model [2] and implemented mainly in C++ [1]. It provides an environment for the development of management applications which hides the details of the underlying management service through object-oriented Application Program Interfaces (APIs) and allows designers and implementors to concentrate in the intelligence to be built in management applications rather than the mechanics of management protocol access. The manager-agent model and the notion of managed objects as abstractions of real resources are used but the separation between managing and managed systems is not strong in engineering terms: a management application can be in both roles and this is particularly true in situations where a management system is decomposed according to a hierarchical logical layered approach. This is exactly the model suggested by the TMN [7] and OSIMIS provides special support to realise management hierarchies.

In fact, OSIMIS was designed from the beginning with the intent to support the integration of existing systems with either proprietary management facilities or different management models. Different methods for the interaction with real managed resources are supported, encompassing loosely coupled resources as it is the case with subordinate agents and management hierarchies. The fact that the OSI model was chosen as the basic management model facilitates the integration of other models, the latter usually being less powerful, as it is the case with the Internet SNMP [6] and most probably with the OSF's DME Advanced Framework [10]. OSIMIS provides already a generic application gateway between CMIS and SNMP [8][9] while a similar approach for the OSF DME may be pursued in the future. The approach of using OSI management on an end-to-end basis for unifying diverse management technologies is in line with the OSI NMF OMNIPoint [11] approach which suggests a multiple technology architecture for future heterogeneous environments.

OSIMIS uses the ISODE (ISO Development Environment) [12] as the underlying OSI communications mechanism but it may also be decoupled from it when the currently ongoing work on supporting the X/Open XOM/XMP/XDS [13] APIs is finalised. The advantage of the ISODE environment though is the provision of services like FTAM and the OSI Directory Service (X.500) (FTAM is suggested as an integral part of a TMN Q3 stack in Q.812 [14]) and the richness of the underlying supported network technologies (X.25, CLNP and also IP) which constitute the majority of currently deployed networks. Interoperation of applications across any of these is possible through Transport Service Bridging [15].

History

OSIMIS started during the ESPRIT-I INCA (Integrated Communications Network Architecture) project as a tool to monitor the OSI transport layer and observe and log the activity of applications such as directories, file transfer and mail systems using it. That version was written in C in a non-object oriented fashion as at the time OSI management standardisation was still in embryonic stage. In the following ESPRIT-II PROOF (Primary Rate OSI Office Facilities) project, the need emerged to manage in an OSI fashion the IP to primary rate ISDN gateway developed in that project. It was when that the idea of building an object-oriented distributed platform with high-level APIs emerged as it became clear that this was the key to reusability and extensibility. The first version of the agent infrastructure was developed then while other components have also been contributed by PROOF.

Work on OSIMIS continued in the RACE-II NEMESYS project where some of the ideas of marrying OSI and ODP principles emerged with concepts like shadow MIBs and transparencies while a hierarchical TMN compliant Service Management system for a simulated ATM network was developed [17]. Currently work on OSIMIS is continuing in the RACE-II NEMESYS successor ICM (Integrated Communications Management) and the ESPRIT-III MIDAS (Management In a Distributed Application and Service environment). Other RACE-II projects such as PREPARE, BAF and GEMA are using it while the first is also a contributor to a small scale. ICM has substantially enhanced it with many new components while important work is still ongoing. Within ICM, OSIMIS has been used in the first phase of the project to realise a hierarchical performance management TMN for a high speed LAN (FDDI) and a simulated ATM network. Different case studies are envisaged for the second phase over a real ATM network - Call Acceptance and Virtual Path Management and also a Virtual Private Network with Frame Relay over ATM [18].

Components and Architecture

OSIMIS as platform comprises the following types of support:

- high level object-oriented APIs realised as libraries
- tools as separate programs supporting the above APIs (compilers/translators)
- generic applications such as browsers, gateways, directory servers etc.
- specific useful management applications

Some of these services are supported by ISODE and these are:

- the OSI Transport (class 0), Session and Presentation protocols, including a lightweight version of the latter that may operate directly over the Internet TCP/IP
- the Association Control, Remote Operations and Reliable Transfer Service Elements (ACSE, ROSE and RTSE) as building blocks for higher level services
- the File Transfer Access and Management (FTAM) and Directory Access Service Element (DASE) built over (some of) the previous ones
- ASN.1 compilers with C language bindings (the *posy/pepy* and *pepsy* tools)
- a Remote Operations stub generator (the *rosy* tool)
- a FTAM service for the UNIX operating system
- a full Directory Service implementation including an extensible Directory Service Agent (DSA) and a set of Directory User Agents (DUAs)
- a transport service bridge allowing interoperability of applications over different network technologies

OSIMIS is built as an environment using ISODE and is mostly implemented in the C++ programming language. The services it offers are:

- an implementation of CMIS/P using the ISODE ACSE, ROSE and ASN.1 tools
- an implementation of the Internet SNMP over the UNIX UDP implementation using the ISODE ASN.1 tools
- high-level ASN.1 support that encapsulates ASN.1 syntaxes in C++ objects
- an ASN.1 meta-compiler which uses the ISODE pepsy compiler to automate to a large extent the generation of syntax C++ objects
- a Coordination mechanism that allows to structure an application in a fully event-driven fashion and can be extended to interwork with similar mechanisms
- a Presentation Support service which is an extension of the coordination mechanism to interwork with X-Windows based mechanisms

- the Generic Managed System (GMS) which is an object-oriented OSI agent engine offering a high level API to implement new managed object classes, a library of generic attributes, notifications and objects and systems management functions
- a compiler for the OSI Guidelines for the Definition of Managed Objects (GDMO) [19] language which complements the GMS by producing C++ stub managed objects covering every syntactic aspect and leaving only behaviour to be implemented
- the Remote MIB high level object-oriented manager API
- a Directory Support service offering application addressing and location transparency services using the ISODE X.500 implementation
- a generic CMIS to SNMP application gateway driven by a translator between SNMP and OSI GDMO MIBs
- a set of generic manager applications (MIB browser and other)
- management agents for the OSI version of the Internet TCP/IP MIB (native version) and the OSI transport protocol

Figure 1. The OSIMIS Layered Architecture and Generic Applications

The OSIMIS services are shown in Figure 1. In the layered part, applications are programs while the rest are building blocks apart from the ASN.1 and GDMO supporting tools. The lower part shows the generic applications provided.

2. The ISO Development Environment

The ISO Development Environment (ISODE) [12] is a platform for the development of OSI services and distributed systems. It provides an upper layer OSI stack that conforms fully to the relevant ISO/CCITT recommendations, including tools for ASN.1 manipulation and remote operations stub generation. Two fundamental OSI applications also come with it, an extensible full Directory Service (X.500) [20] and File Transfer (FTAM) [21] implementations. ISODE is implemented in the C programming language [22] and runs on most versions of the UNIX operating system.

ISODE does not provide any network and lower layer protocols e.g. X.25, CLNP, but relies on implementations for UNIX-based workstations which are accessible through the kernel interface. The upper layer protocols realised are the transport, session and presentation protocols of the OSI 7-layer model. Application layer Service Elements (ASEs) are also provided as building blocks for higher level services, these being the Association Control, Remote Operations and Reliable Transfer Service Elements (ACSE, ROSE and RTSE). These, in conjunction to the ASN.1 support, are used to implement higher level services. A special lightweight presentation layer is also provided that runs directly on top of TCP; this may be used for the CMOT (CMIP over TCP) [16] stack. In engineering terms, the ISODE stack is a set of libraries linked with applications using it.

Figure 2. The ISODE OSI Stack

Remote operations in OSI are formally specified through the ROSE ASN.1 template which is functionally similar to Interface Definition Languages (IDL) of distributed systems platforms. Base i.e. non-template ASN.1 is used to specify the parameters of operations, results and errors. A compiler for that template named *rosy* is provided and automates the generation of stub remote operations in much the same way as IDL compilers in ODP-based distributed systems platforms. What is different to those is that there is no trader for exporting services and binding clients to servers. As discussed in section 8.4.1, the OSI Management model together with the Directory Service may be used to provide such facilities.

Base ASN.1 manipulation is also very important. The ISODE approach for a programmatic interface (API) relies in a fundamental abstraction known as Presentation Element (PE). This is a generic C structure capable of describing in a recursive manner any ASN.1 data type. An ASN.1 compiler known as *pepsy* is provided with C language bindings, which produces concrete representations i.e. C structures corresponding to the ASN.1 types and also encode/decode routines that convert those to PEs and back. The presentation layer converts PEs to a data stream according to the encoding rules (e.g. BER) and the opposite. It should be noted that X/Open has defined an API for ASN.1 manipulation known as XOM

[13] which, though similar in principle to that of ISODE, is syntactically very different. Translations between the two should be possible and such approaches are being investigated. A depiction of the upper layer ISODE stack is shown in figure 2.

One of the most important concepts pioneered in ISODE is that of interworking over different lower layer protocol stacks which is realised through Transport Service bridging (TS-bridge) [15]. ISODE provides an implementation of the ISO Transport Protocol (TP) class 0 over X.25 or even over the Internet TCP/IP using the RFC1006 method [23], in which TCP is treated as a reliable network service. The ISODE session protocol may also run over the ISO TP class 4 and the Connectionless Network Protocol (CLNP). Transport service bridges, which are simple relaying applications similar to Interworking Units (ITUs), may be used to link subnetworks of all these different *communities* and to provide end-to-end interoperability hiding the heterogeneity of the underlying network technology. The combinations mentioned before constitute the vast majority of currently deployed networks.

3. Management Protocol and High-Level Abstract Syntax Support

OSIMIS is based on the OSI management model as the means for end-to-end management and as such it implements the OSI Common Management Information Service/Protocol (CMIS/P). This is implemented as a C library and uses the ISODE ACSE and ROSE and its ASN.1 support. Every request and response CMIS primitive is realised through a procedure call. Indications and confirmations are realised through a single *"wait"* procedure call. Associations are represented as communication endpoints (file descriptors) and operating system calls e.g. the Berkeley UNIX select(2) can be used for multiplexing them to realise event-driven policies.

The OSIMIS CMIS API is known as the MSAP API, standing for Management Service Access Point. It was conceived much before standard APIs such as the X/Open XMP were specified and as such it does not conform to the latter. Having been designed specifically for CMIS and not for both CMIS and SNMP as the XMP one, it hides more information and may result in more efficient implementations. The reason this is a procedural C object-based and not a fully object-oriented implementation is to conform to the ISODE style, the trend of industry APIs and to be easily integrated in diverse environments. Higher-level object-oriented abstractions that encapsulate this functionality and add much more can be designed and built as explained in section 6.

OSIMIS offers as well an implementation of the Internet SNMPv1 and v2 which is used by the generic application gateway between the two. This uses the UNIX implementation of the Internet UDP and the ISODE ASN.1 support and is implemented in much the same fashion as CMIS/P, without conforming to XMP. Applications using CMIS need to manipulate ASN.1 types for the CMIS managed object attribute values, action, error parameters and notifications. The API for ASN.1 manipulation in ISODE is different to the X/Open XOM. Migration to XOM/XMP is though possible through thin conversion layers so that the upper layer OSIMIS services are not affected. The XMP part of this conversion is being implemented in the ESPRIT MIDAS while the XOM part will be implemented by the RACE ICM project.

Regarding ASN.1 manipulation, it is up to an application to encode and decode the above values as this adds to its dynamic nature by allowing late bindings of types to values and graceful handling of error conditions. From a distributed programming point of view this is unacceptable and OSIMIS provides an mechanism to support high-level object-oriented ASN.1 manipulation, shielding the programmer from details and enabling distributed programming using simply C++ objects as data types. This is achieved by using polymorphism to encapsulate behaviour in the data types determining how encoding and decoding should be performed through an ASN.1 meta-compiler which produces C++ classes for each type. Encode, decode, parse, print and compare methods are produced together with a get-next-element one for multi-valued types (ASN.1 SET OF or SEQUENCE OF). Finally, the very important ANY DEFINED BY construct is automatically supported through a table driven approach, mapping types to syntaxes. An example of instances of such object-oriented ASN.1 manipulation is:

```
// a value in the heap
int* i = new int; *i = 5;

// three different styles for the value
AsnType* i1 = new Integer(i);              // specific syntax class
AsnType* i2 = new AnyType("integer", "5"); // general class - sntx table driven
AsnType* i3 = new AnyType("integer", i);   //   ...

// Attribute Value Assertion (AVA) - type:value
AVA* a1 = new AVA("pktsSent", i);          // ANY DEFINED BY - OID table driven
```

The above ASN.1 approach is used throughout in higher level OSIMIS services such as the GMS (section 5) and high-level manager APIs (section 6).

4. Application Coordination Support

Management and, more generally, distributed applications have complex needs in terms of handling external input. Management applications have additional needs of internal alarm mechanisms for arranging periodic tasks in real time (polling etc.) Furthermore, some applications may need to be integrated with Graphical User Interface (GUI) technologies which have their own mechanisms for handling data from the keyboard and mouse. In this context, the term application assumes one process in operating systems terms.

There are in general different techniques to organise an application for handling both external and internal events. The organisation needs to be event driven though so that no resources are used when the system is idle. The two major techniques are:

a. use a single-threaded execution paradigm

b. use a multi-threaded one

In the first, external communications should follow an asynchronous model as waiting for a result of a remote operation in a synchronous fashion will block the whole system. Of course, a common mechanism is needed for all the external listening and demultiplexing of the incoming data and this is a part of what the OSIMIS Application Coordination Support provides. In the second, many threads of control can be executing simultaneously (in a pseudo-parallel fashion) within the same process, which means that blocking on an external result is allowed. This is the style of organisation distributed systems platforms such as the OSF's DCE and ANSA suggest, as they are based on RPC which is inherently synchronous.

The advantage of the first mechanism is that it is supported by most operating systems and, as such, it is lightweight and efficient while its drawbacks are that it introduces state for handling asynchronous remote operation results. The second mechanism allows more natural programming in a stateless fashion with respect to remote operations but it requires internal locking mechanisms and re-entrant code. In addition, such mechanisms are not yet commonly supported by operating systems and as such are not very efficient. An additional problem in organising a complex application concerns the handling of internal timer alarms: most operating systems do not *"stack"* them i.e. there can only be one alarm pending for each process. This means that a common mechanism is needed to ensure the correct usage of the underlying mechanism.

OSIMIS provides an object-oriented infrastructure in C++ which allows to organise an application in a fully event-driven fashion and a single- threaded execution paradigm, where every external or internal event is serialised and taken to completion on a *"first-come-first-served"* basis. This mechanism allows the easy integration of additional external sources of input or timer alarms and it is realised by two C++ classes: the Coordinator and the Knowledge Source (KS). There should always be one instance of the Coordinator or any derived class in the application while the Knowledge Source is an abstract class that allows to use the coordinator services and integrate external sources of input or timer alarms. All external events and timer alarms are controlled by the coordinator and the model is depicted in Figure 3.

Figure 3. The OSIMIS Coordination Mechanism

This coordinating mechanism is designed in such a way as to allow integration with other systems coordination mechanisms. This is achieved through special classes coordinator derived classes which will interwork with the particular mechanism. This is achieved by still controlling the sources of input and timer alarms of the OSIMIS KSs but instead of performing the central listening, these are passed to the other system's coordination mechanism which becomes the central one.

This is needed for OSIMIS agents which wish to receive association requests since ISODE imposes its own listening mechanism which hides the Presentation Service Access Point (PSAP) on which new ACSE associations are accepted. As such, the IsodeCoordinator class is needed in conjunction with the IsodeAgent which is an abstract KS able to receive association requests. A similar mechanism is needed for Graphical User Interface technologies which have their own coordination mechanisms. In this case, simply a new special coordinator class is needed for each of them. At the moment, the X-Windows Motif, the TCL/TK interpreted language and the InterViews graphical object library are integrated. The inheritance hierarchy for all these classes is shown in Figure 4.

```
             Coord                                KS
               |                                  |
       ------------------                         |
       |        |         |                       |
   IsodeCoord XCoord TclCoord               IsodeAgent
```

5. The Generic Managed System

The Generic Managed System (GMS) provides support for building agents that offer the full functionality of the OSI management model, including scoping, filtering, access control, linked replies and cancel-get. OSIMIS supports fully the Object Management [24], Event Reporting [25] and Log Control [26] Systems Management Functions (SMFs), the *qualityofServiceAlarm* notification of the Alarm Reporting one and partly the Access Control [27], Metric [28] and Summarisation [29] objects. In conjunction with the GDMO compiler it offers a very high level API for the integration of new managed object classes where only semantic aspects (*behaviour*) need to be implemented. It also offers different methods of access to the associated real resources, including proxy mechanisms, based on the Coordination mechanism.

The Generic Managed System is built using the coordination and high level ASN.1 support infrastructure and most of its facilities are provided by three C++ classes which interact with each other:

- the CMISAgent, which provides OSI agent facilities
- the MO which is the abstract class providing generic managed object support
- the MOClassInfo which is a meta-class for a managed object class

The GMS library contains also generic attribute types such as counter, gauge, counterThreshold, gaugeThreshold and tideMark and specific attributes and objects as in the Definition of Management Information (DMI) [30], which relate to the SMFs. The object-oriented internal structure of a managed system built using the GMS in terms of interacting object instances is shown in Figure 5.

Figure 4. The OSIMIS GMS Object-Oriented Architecture

5.1 The CMIS agent

The CMISAgent is a specialised knowledge source, more specifically a specific IsodeAgent, as it has to accept management associations. There is always one only instance of this class for every application in agent role. Its functions are to accept or not associations according to authentication information, check the validity of operation parameters, find the base object for the operation, apply scoping and filtering, check if atomic synchronisation can be enforced, check access control rights and then apply the operation on the target managed object(s) and return the result(s)/error(s).

There is a very well defined interface between this class and the generic MO one which is at present synchronous only: a method call should always return with a result e.g. attribute values or error. This means that managed objects which mirror loosely coupled real resources and exercise an *"access-upon-external-request"* regime will have to access the real resource in a synchronous fashion which will result in the application blocking until the result is received. This is only a problem if another request is waiting to be served or if many objects are accessed in one request through scoping. Threads would be a solution but the first approach will be a GMS internal asynchronous API which is currently being designed. It is noted that the CMISAgent to MO interface is bidirectional as managed objects emit notifications which may be converted to event reports and passed to the agent.

5.2 Object Classes and Instances

Every specific managed object class needs access to information common to the class which is independent of all instances and common to all of them. This information is related to which are the attributes, actions and notifications for the class, initial and default attribute values, "template" ASN.1 objects for manipulating action and notification values, integer tags associated to the object identifiers etc. This leads to the introduction of a common meta-class for all the managed object classes, the MOClassInfo. The inheritance tree is internally represented by instances of this class linked in a tree fashion as shown in Figure 5.

Specific managed object classes are simply realised by equivalent C++ classes produced by the GDMO compiler and augmented manually with behaviour. Through access to meta-class information requests are first checked for correctness and authorisation before the behaviour code that interacts with the real resource is invoked. Behaviour is implemented through a set of polymorphic methods which may be redefined to model the associated real resource. Managed object instances are linked internally in a tree mirroring the containment relationships. Scoping is relegated to a tree search while special care is taken to make sure the tree reflects reality when accessed. Filtering is provided through compare methods of the attributes which are simply the C++ syntax objects or derived classes when behaviour is coded at the attribute level.

5.3 Real Resource Access

There are three possible types of interaction between the managed object and the associated resource with respect to CMIS Get requests:

1. access upon external request
2. "cache-ahead" through periodic polling
3. update through asynchronous reports

The first one means that no activity is incurred when no manager accesses the agent but cannot support notifications. In the second requests are responded quickly, especially with respect to loosely coupled resources, but timeliness of information may be slightly affected. Finally the third one is good but only if it can be tailored so that there is no unnecessary overhead when the agent is idle.

The GMS offers support for all of them through the coordination mechanism. When asynchronous reports from a resource are expected or asynchronous results to requests, it is likely that a separate object will be needed to demultiplex the incoming information and deliver it to the appropriate managed object. It should noted here that an asynchronous interface to real resources driven by external CMIS requests is not currently supported as this requires an internal asynchronous interface between the agent and the managed objects. These objects are usually referred to an Internal Communications Controllers (ICCs) and are essentially specialised knowledge sources.

The internal organisation of a GMS-based agent in terms of the major interacting object instances is shown in Figure 5. Note that the instances shown are only the major ones defining the internal flow of control. The application’s intelligence may be realised through a whole lot of other objects which are application specific. Note also that the OSI stack is essentially encapsulated by the CMISAgent object.

5.4 Systems Management Functions

As already stated, OSIMIS supports the most important of the systems management functions. As far as the GMS is concerned, these functions are realised as special managed objects, generic attribute and notification types which can be simply instantiated or invoked. This is the case for example with the alarm reporting, metric and summarisation objects. In other cases, the GMS knows the semantics of these classes and uses them accordingly e.g. in access control and event and log control. Notifications can be emitted through a special method call and all the subsequent notification processing is carried out by the GMS in a transparent fashion to application code. In the case of object management, the generated code by the GDMO compiler together with the GMS hide completely the emission of object creation and deletion notifications and the attribute change one when something is changed through CMIS.

Log control is realised simply through managed object persistency which is a general property of all the OSIMIS managed objects. This is implemented using the GNU version of the UNIX DBM database management system and relies on object instance encoding using ASN.1 and the OSI Basic Encoding Rules to serialise the attribute values. Any object can be persistent so that its values are retained between different incarnations of an agent application. At start-up time, an agent looks for any logs or other persistent objects and simply arranges its management information tree accordingly.

6. Generic High-Level Manager Support

Programming manager applications using the CMIS API can be tedious. Higher object-oriented abstractions can be built on top of the CMIS services and such approaches were initially investigated in the RACE-I NEMESYS project while work in this area was taken much further in the RACE-II ICM project.

6.1 The Remote MIB Support Service

The Remote MIB (RMIB) support service offers a higher level API which provides the abstraction of an association object. This handles association establishment and release, hides object identifiers through friendly names, hides ASN.1 manipulation using the high-level ASN.1 support, hides the complexity of CMIS distinguished names and filters through a string-based notation, assembles linked replies, provides a high level interface to event reporting which hides the manipulation of event discriminators and finally provides error handling at different levels. There is also a low level interface for applications that do not want this friendliness and the performance cost it entails but they still need the high-level mechanisms for event reporting and linked replies.

In the RMIB API there are two basic C++ classes involved: the RMIBAgent which is essentially the association object (a specialised KS in OSIMIS terms) and the RMIBManager abstract class which provides call-backs for asynchronous services offered by the RMIBAgent. While event reports are inherently asynchronous, manager to agent requests can be both: synchronous, in an RPC like fashion, or asynchronous. In the latter case linked replies could be all assembled first or passed to the specialised RMIBManager one by one. It should be noted that in the case of the synchronous API the whole application blocks until the results and/or errors are received while this is not the case with the asynchronous API. The introduction of threads or coroutines will obviate the use of the asynchronous API for reasons other than event reporting or a one-by-one delivery mechanism for linked replies. The RMIB model is shown in Figure 6.

6.2 The Shadow MIB Support Service

While the RMIB infrastructure offers a much higher level facility than a raw CMIS API such as the OSIMIS MSAP one or X/Open’s XOM/XMP, its nature is closely linked to that of CMIS apart from the fact that it hides the manipulation of event forwarding discriminators to effect event reporting. Though this facility is perfectly adequate for even complex managing applications as it offers the full CMIS power (scoping, filtering etc.), simpler higher-level approaches could be very useful for rapid prototyping.

One such facility is provided by the Shadow MIB SMIB) support service, which offers the abstraction of objects in local address space, "shadowing" the real managed objects handled by remote agents. The real advantages of such an approach are twofold: first, the API could be less CMIS-like for accessing the local objects since parameters such as distinguished names, scoping etc. can be just replaced by pointers in local address space. Second, the existence of images of MOs as local shadow objects can be used to cache information and optimise access to the remote agents. The cacheing mechanism could be controlled by local application objects, tailoring it according to the nature of the application in hand in conjunction with shared management knowledge regarding the nature of the remote MIBs.

Issues related to the nature of such an API are currently investigated in the ICM project. The model and supporting C++ classes are very similar to the RMIB ones and are illustrated in Figure 6.

Figure 5. The RMIB and SMIB Manager Support Models

6.3 Scripting CMIS-Based Languages

Both the RMIB and SMIB support services are based on a compiled model while interpreted models are more suitable for quick prototyping, especially when similar mechanisms for Graphical User Interfaces are available. Such mechanisms currently exist e.g. the TCL/TK language/widget set or the SPOKE object-oriented environment and these are used in the RACE ICM project as technologies to support GUI construction.

Combining them to a CMIS-like interpreted scripting language can lead to a very versatile infrastructure for the rapid prototyping of applications with graphical user interfaces. Such languages are currently being investigated in the ICM and other projects.

7. Directory Support Services and Distribution

Management applications need to address each other in a distributed environment. The OSI Directory Service (X.500) provides the means for storing information to make this possible. It essentially provides the a highly distributed hierarchical information store with high performance access characteristics, so it is suitable for storing information related to the location and capabilities of management applications.

The Directory Service model structures information in an object-oriented hierarchical fashion similar to that of OSI management. Information objects can be created, have their attributes accessed (read or written) and deleted. Access can involve complex assertions through filtering as with CMIP. This object-oriented information store can be highly distributed over physically separate entities known as Directory Service Agents (DSAs). These communicate with each other through a special protocol and requests for information a DSA does not hold can be "chained" to all the other DSAs until the information is found. This information can be accessed through Directory User Agents (DUAs) which talk to the local domain DSA through the Directory Access Protocol (DAP) while chaining guarantees the search of the global information store.

This model is very powerful and resembles a lot to that of OSI management. From an information modelling perspective, the latter is a superset of the X.500 one and could be used to much the same effect. It is the chaining facility though that distinguishes the two and makes X.500 more suitable as a global information store. Current research tries to unify the two models, considering all the Management Information Trees of applications as extensions of the global Directory Information Tree.

Directory Services can be used in OSIMIS for application addressing in two different styles: the first resolving Application Entity Titles (AETs) to Presentation Addresses (PSAPs) in a static fashion with the second introducing dynamic "location transparency" services as in distributed systems platforms. OSIMIS applications may as well operate without the Directory Service: in this latter case, a flat file known as "isoentities" should exist at each site where management applications need addressing information. This maps statically AETs to PSAPs and should be administered in a fashion that guarantees global consistency, but this is exactly the purpose of services like X.500!

In the first level of X.500 usage, the static information residing normally in the isoentities file is converted into directory objects and stored in the directory (the ISODE QUIPU X.500 implementation provides a utility to do this off-line. This information becomes then globally accessible while central administration and consistency maintenance become fairly simple. This approach is adequate for fairly static environments where changes to the location of applications are infrequent. For more dynamic environments like the TMN where distributed applications in the higher layers may often be moved for convenience, resilience, replication etc., a highly flexible solution is needed. This is provided in the form of location transparency services, wherever these are appropriate. It should be noted that these services may not be appropriate for the lowest TMN layer (Network Element), as the same application e.g. an agent for an ATM switch would run at multiple sites with topology information regarding the location used to access its services.

Location transparency is implemented through special directory objects holding location, state and capability information of management applications. The latter register with it at start-up time and provide information of their location and capabilities while they de-register when they exit. Applications that wish to contact another one for which the know its logical name (AET), they contact the directory through a generic "broker" module they contain and may obtain one or more locations where this application runs. Further criteria e.g. location may be used to contact the right one. Another level of indirection can be used when it is not the name of an application known in advance but the name of a resource. A special directory information model has been devised that allows this mapping by following "pointers" i.e. Distinguished Names that provide this mapping. Complex assertions using the directory access filtering mechanism can implemented to allow the specification of a set of criteria for the service or object sought.

8. Applications

OSIMIS is a development environment (i.e. a platform) for highly distributed management systems. As such it encompasses libraries providing APIs that can be used to realise applications, some of the APIs supported by stand-alone programs as the ASN.1 and GDMO compilers, and also management applications that are either generic tools or useful examples using the infrastructure.

There are two types of generic applications: semantic-free manager ones that may operate on any MIB without changes and agent generic gateways for other management models. OSIMIS provides a set of generic managers, graphical or command-line based, which provide the full power of CMIS and a generic application gateway between CMIS/P and the Internet SNMP.

8.1 Generic Managers

There is a class of applications which are semantic-free and these are usually referred to as MIB browsers as they allow one to move around in a management information tree, retrieve and alter attribute values, perform actions and create and delete managed objects. OSIMIS provides a MIB browser with a Graphical User Interface based on the InterViews X-Windows C++ graphical object library. This allows only the read and write operations currently and also provides a monitor facility to observe an object periodically and report changes. It is soon going to be extended with Action, Create and Delete support and also the capability to receive event reports and even monitor objects through event reports. Its GUI support engine is also going to be rewritten using the TCL/TK interpreted language / widget set.

OSIMIS provides also a set of programs that operate from the command line and realise the full set of CMIS operations. These are going to be combined together in a "management shell". There is also an event sink application that can be used to receive event reports according to specified criteria. Both the MIB browser and these command line programs owe their genericity to the generic CMIS facilities (empty local distinguished name {} for the top MIB object, the localClass facility and scoping) and the manipulation of the ANY DEFINED BY ASN.1 syntax through the table driven approach described in section 3.

8.2 The Generic CMIS/SNMP Application Gateway

The current industry standard for network element management is the Internet SNMP, which is a simplified version of the OSI CMIP and the same holds for the relevant information models, the OSI being fully object-oriented while the SNMP supporting a simple remote debugging paradigm. Application gateways between them are possible for specific Management Information Bases (MIBs) - in fact OSIMIS comprised such a gateway for the OSI version of the Internet MIB-II for the TCP/IP protocol family.

This non-generic approach though requires the realisation of a new gateway for every new MIB and as such is not a scalable solution. A better approach is to define a generic set of rules for the conversion between the two information models and have a generic gateway which may operate for any MIB. Work for standards in this area has been driven by the Network Management Forum (NMF) while the RACE ICM contributed actively to them and also built a generic application gateway based on OSIMIS [8] [9].

This work involves a translator between Internet MIBs to equivalent GDMO ones and a special back-end for the GDMO compiler which will produces run-time support for the generic gateway. That way the handling of any current or future MIBs will be possible without the need to change a single line of code. It should be added that the generic gateway works with SNMP version 1 but it will be soon extended to also cover SNMP version 2. The current approach for the gateway is stateless but the design is such that allows to introduce easily future stateful optimisations in order to reduce SNMP managed systems access.

8.3 Generally Useful Agents

There are two useful agents implementing specific MIBs which come with OSIMIS. The first implements a native (non-proxy) implementation of the OSI version of the TCP/IP MIB as in the RFC 1214 [31]. This is going to be updated soon to reflect the IIMC work [32] as the RFC 1214 is now obsolete. This particular application is very important as TCP/IP is the dominant networking technology today, especially in LANs and MANs and can be used to find out the utilisation of interfaces, manage addresses and routes in workstations etc.

The second application contains a non-standard implementation of the MIB for the OSI Transport Protocol (TP) which will be brought in accordance to the relevant standards in the future. This manages in particular the TP implementation of the ISODE stack and is very useful to see the activity of applications using ISODE e.g. DSAs, MTAs, transport bridges and even, though rather incestuous, of management applications themselves!

8.4 Other Potential Uses and Applications

Up to now it has been assumed that OSIMIS will be used as a management systems platform. Though this is the reason it was conceived for, the power and generality of the OSI management model, on which OSIMIS is based, together with the abstractions OSIMIS provides and make it suitable for other potential usages outside the management systems realm.

8.4.1 OSIMIS as a General Distributed Systems Platform

The facilities OSIMIS provides make it suitable as a general vehicle for building distributed systems: the Action CMIS primitive is essentially a general method on an object which can have any parameters and return results or errors. This object is contained within an application, together possibly with other objects and its services may be accessed by objects in other applications in a distributed manner. The Directory Support Services enable to one to find out which objects operate in a domain, what services they provide, which application contains them and finally use their services in a location transparent fashion. In combination to the high level object access mechanisms (RMIB / SMIB), it is possible to build distributed systems quickly and efficiently.

Realising a simple time-of-the-day server requires no more than 10 lines of C++ code, using the GMS and GDMO compiler. Accessing all these objects in a particular environment using the location transparency and the RMIB support services and printing the time requires about 20 lines of C++ code. Finally, the performance of OSIMIS-based systems is generally high as various experiments have shown.

8.4.2 OSIMIS as an Object-Oriented Distributed Database Mechanism

OSIMIS provides managed object persistency and in particular the facility to log information as records (special managed objects) and to fully control the behaviour of the logs containing them. The OSI management service offers the possibility to create, delete and retrieve logs and log records and also sophisticated searching facilities through scoping and filtering. Nothing prevents one to create special log record classes which could be created/deleted and manipulated through the management protocol - standard eventLogRecords are created within the managed system as a result of notifications and criteria set by managers.

Despite the fact that this was not the purpose of the OSI management model, its generality and richness make it suitable as a general distributed object-oriented database mechanism. OSIMIS does not provide yet transaction support which is very important for such usages. When these are implemented, the existing infrastructure will make possible to use OSIMIS for storing information in a distributed fashion while providing sophisticated authentication and access control mechanisms. It should be noted that transactional services are particularly useful in the higher TMN layers where more static information as e.g. for services, customers etc. is stored.

8.5 The Generic OSIMIS-Based TMN Operations System

Every OSIMIS-based application has an object-oriented structure dictated by the infrastructure it uses. A TMN Operations System (OS) acts in both agent and manager roles, as agent to superior or peer OSs and as manager to peer or subordinate ones. The object-oriented structure of the generic OSIMIS-based OS is shown in Figure 7. This comprises the GMS infrastructure for the agent part, simplified for the purpose of this depiction, possibly many RMIB/SMIB parts depending on the number of remote OSs in agent roles and the directory support object to address the latter.

Figure 6. The Generic OSIMIS-based TMN Operations System

9. Epilogue

OSIMIS has proved the feasibility of OSI management and especially its suitability of its object-oriented concepts as the basis for higher-level abstractions which harness its power and hide its complexity. It has also shown that a management platform can be much more than a raw management protocol API together with sophisticated GUI support which is that of most commercial offerings. In an environment like the TMN where a hierarchical organisation of management applications is paramount, object-oriented agent support similar to that of the GMS and the associated tools and functions is fundamental together with the ability to support the construction of proxy systems (Q-Adaptors). Higher level manager support is also important to hide the complexity of CMIS services and allow the rapid but efficient systems realisation.

OSIMIS has also shown that an RPC mechanism together with an Interface Definition Language (IDL) and a trader is not the only model to build distributed systems and, in fact, it is not adequate for management systems. Broking services like the ones traders offer can be built using the OSI Directory (X.500) and Management (X.700) models while at the same time management applications conform fully to the ISO/CCITT management standards. The importance of standard management models and protocols as the basis for experimental systems had been neglected in RACE-I in favour of approaches based on models for distributed systems. Projects like the RACE-I NEMESYS and RACE-II ICM have shown that Open Distributed Processing (ODP) and Open Systems Interconnection principles (OSI) can harmonically coexist, the former using the latter as means for the realisation of its principles.

Finally, as we live in a multi-model and protocol world, it is unwise to assume existing investment will be simply thrown away to conform to a new model. In the management world there are two prevalent solutions, OSI and Internet, while a third one, the OSF's ODP-biased DME Advanced Framework threatens to become a de-facto industrial standard due to the fact it comes as a software platform rather than a pile of documents. All these three solutions will have to co-exist in the years to come and the basis for their integration through generic application gateways should be the most powerful of the three in order to ensure translation without loss of semantics. Though the details of the DME AF are not yet known, it is unlikely to be more powerful than the OSI one in terms of both protocol operations and information model expressiveness. As such, the choice of the OSI model as the basis of OSIMIS together with the facilities it provides for the construction of generic proxy systems, of which the CMIS to SNMP application gateway is a tangible proof, justify its characterisation as a platform for multiple technology, integrated management systems.

Future Work

There is currently a lot of ongoing work in the ICM and MIDAS projects in enhancing OSIMIS with other services. Some of the envisaged work is listed below:

- an asynchronous GMS-internal API to overcome the current performance problem for multiple simultaneous CMIS requests for loosely coupled resources
- experiment with both coroutine and thread mechanisms as a solution obviating the use of asynchronous APIs
- enhance the access control implementation and provide related systems management functions such as security alarm reporting and security audit trail
- design further and implement the shadow MIB manager support API
- design and implement an interpreted scripting language for CMIS
- implement the repertoire and discovery shared management knowledge objects for fully dynamic managers
- work further on the location and other ODP transparencies such as migration, resilience etc.
- complete the XOM/XMP/XDS implementation which will make OSIMIS independent of the underlying stack (currently operates only on ISODE)
- research on ways of realising policies as managed objects, on domains and on the formal definition of behaviour
- investigate generic gateways between OSI and other models such as the OSF's DME Advanced Framework
- and a lot more...

Acknowledgements

Many people have contributed in the development of OSIMIS. The author architected and implemented the core of the system and has been influencing its direction and evolution in the RACE NEMESYS and ICM projects. Graham Knight of UCL has also influenced the original concept, designed the security service and the XMP compatibility libraries, implemented part of the latter and has being influencing its evolution in the ESPRIT PROOF and MIDAS projects.

Saleem Bhatti of UCL designed and implemented the log control and managed object persistency service, the first version of the GDMO code generation scripts and the OSI version of the Internet MIB. He has also been involved in the design of the GMS providing valuable feedback and in the initial design of the CMIS/SNMP generic gateway. James Cowan of UCL designed and implemented the generic MIB browser, the first version of the RMIB service and the excellent platform-independent GDMO compiler. Kevin McCarthy of UCL implemented the first non-generic version of the CMIS->SNMP gateway for the Internet MIB and designed and implemented the major part of the generic version. Luca Deri of UCL implemented the first version of the object-oriented ASN.1 meta-compiler and the security service.

Thurain Tin of Unipro designed and implemented the full RMIB service. Costas Stathopoulos and David Griffin of ICS designed the location transparency mechanism which was implemented by the former. Jim Reilly of VTT implemented the metric objects, the excellent Internet to GDMO MIB translator and together with Juha Coivisto, also of VTT, provided general valuable feedback. Xavier Pandolfi of Alcatel ISR cooperated in the implementation of the presentation support service aspect of the coordination mechanism. Neuman De Souza of DNAC cooperated in the design and implementation of the generic CMIS/SNMP gateway. Andy Carr of Cray Communications has been providing useful input within the ICM project and is currently designing and implementing the shadow MIB service.

REFERENCES

[1] Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley, Reading, MA, 1986

[2] ISO/IS 10040, *Information Technology - Open Systems Interconnection - Systems Management Overview* July 1991

[3] ISO/IS 9595, *Information Technology - Open Systems Interconnection - Common Management Information Service Definition, Version 2*, July 1991

[4] ISO/IS 9595, *Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification, Version 2*, July 1991

[5] ISO/IS 10165-4, *Information Technology - Structure of Management Information - Part 1: Management Information Model*, August 1991

[6] J. Case, M. Fedor, M. Schoffstall, J. Davin, *A Simple Network Management Protocol (SNMP)*, RFC 1157, May 1990

[7] CCITT M.3010, *Principles for a Telecommunications Management Network*, Working Party IV, Report 28, December 1991

[8] G. Pavlou, S.N. Bhatti, G. Knight, *Automating the OSI to Internet Management Conversion Through the Use of an Object-Oriented Platform*, IFIP Conference on LAN and MAN Management, Paris, April 1993

[9] J.N. DeSouza, K. McCarthy, G. Pavlou, N. Agoulmine, *CMIP to SNMP Translation Through Application Gateways Using OSIMIS/ISODE*, RACE Conference on Intelligence in Broadband Services and Networks, Paris, November 1993

[10] Open Software Foundation, *Distributed Management Environment (DME)* DME TECHSEM0 071 592, 1993

[11] Network Management Forum, *Open Management Interoperability Point (OMNIpoint) 1.0*, 1993

[12] M.T. Rose, J.P. Onions, C.J. Robbins, *The ISO Development Environment User's Manual Version 7.0*, PSI Inc / X-Tel Services Ltd, July 1991

[13] X/Open, *OSI-Abstract-Data Manipulation, Management Protocols and Directory Service API Specification*, January 1992

[14] CCITT Q.812, *Upper Layer Protocol Profiles for the Q3 Interface*, Working Group XI, Report 253, 1993

[15] J.P. Onions, *Trasnport Bridging*, Published in Interworking: Research and Experience, Vol. 1, pp 27-34, John Wiley & Sons

[16] L. Besaw, B. Handspicker, L. LaBarre, U. Warrier, *The Common Management Information Services and Protocols for the Internet*, RFC 1189, October 1990

[17] G. Pavlou, A. Mann, U. Harksen, *Experience of Modelling and Implementing a Quality of Service Management System*, Published in The Management of Telecommunications Networks, pp 109-120, Ellis Horwood, 1993

[18] ICM, D. Griffin (editor), *Deliverable 5: Revised TMN Architecture and Case Studies*, October 1993

[19] ISO/IS 10165-4, *Information Technology - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects*, August 1991

[20] ISO/IS 9594-1, *Information Processing - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Service*, 1988

[21] ISO/IS 8571, *Information Processing - Open Systems Interconnection - File Transfer, Access and Management*, 1988

[22] B.W. Kernigham, D.M. Ritchie, *The C Programming Language*, Prentice-Hall, New Jersey, 1978

[23] D. Cass, M. Rose*ISO transport services on top of TCP*, RFC 1006, May 1987

[24] ISO/IEC 10164-5, *Information Technology - Open Systems Interconnection - Systems Management - Part 1: Object Management Function*, August 1991

[25] ISO/IEC 10164-5, *Information Technology - Open Systems Interconnection - Systems Management - Part 5: Event Report Management Function*, August 1991

[26] ISO/IEC 10164-6, *Information Technology - Open Systems Interconnection - Systems Management - Part 6: Log Control Function*, June 1991

[27] ISO/IEC 10164-6, *Information Technology - Open Systems Interconnection - Systems Management - Part 9: Objects and Attributes for Access Control*, June 1991

[28] ISO/IEC 10164-6, *Information Technology - Open Systems Interconnection - Systems Management - Part 11: Metric Objects and Attributes*, June 1991

[29] ISO/IEC 10164-6, *Information Technology - Open Systems Interconnection - Systems Management - Part 13: Summarisation Function*, June 1991

[30] ISO/IEC 10165-2, *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 2: Definition of Management Information*, January 1992

[31] L. LaBarre, *OSI Internet Management: Management Information Base*, RFC 1214, April 1990

[32] L. LaBarre, *ISO/CCITT and Internet Management Coexistence (IIMC): Translation of Internet MIBs to ISO/CCITT GDMO MIBs*, Internet Draft, 1993

CONTENTS

LIST OF FIGURES