The OSIMIS Platform: Making OSI Management Simple

George Pavlou, Kevin McCarthy, Saleem Bhatti, Graham Knight
Department of Computer Science, University College London, Gower Street, London, WC1E 6BT, UK
tel: +44 71 380 7215 fax: +44 71 387 1397
e-mail: g.pavlou k.mccarthy s.bhatti g.knight @cs.ucl.ac.uk

Abstract

The OSIMIS (OSI Management Information Service) platform provides the foundation for the quick, efficient and easy construction of complex management systems. It is an object-oriented development environment in C++ [Strou] based on the OSI Management Model [X701] that hides the underlying protocol complexity (CMIS/P) and harnesses the power and expressiveness of the associated information model [X722] through simple to use Application Program Interfaces (APIs). 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 [SNMP] capable network elements. OSIMIS supports particularly well a hierarchical management organisation through hybrid manager-agent applications and may embrace a number of diverse technologies through proxy systems. This paper explains the OSIMIS components, architecture, philosophy and direction.

Keywords

Network, Systems, Application Management, Distributed Systems, Platform, API

1 INTRODUCTION AND OVERVIEW

OSIMIS is an object-oriented management platform based on the OSI model [X701] and implemented mainly in C++ [Strou]. 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 on the intelligence to be built into management applications rather than the mechanics of management service/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.

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 is the case with the Internet SNMP [SNMP]. OSIMIS provides already a generic application gateway between CMIS and SNMP [Pav93a] while a similar approach for integrating OSI management and the OMG CORBA framework [OMG] may be pursued in the future.

OSIMIS uses the ISODE (ISO Development Environment) [ISODE] as the underlying OSI communications mechanism but it may also be decoupled from it through the XOM/XMP [XOpen] management API. The advantage of the ISODE environment though is the provision of services like FTAM and a full implementation of the OSI Directory Service (X.500) which are essential in complex management environments. Also a number of underlying network technologies are supported, namely X.25, CLNP and also TCP/IP through the RFC1006 method. These constitute the majority of currently deployed networks while interoperation of applications across any of these is possible through Transport Service Bridging.

OSIMIS has been and is still being developed in a number of European research projects, namely the ESPRIT INCA, PROOF and MIDAS and the RACE NEMESYS and ICM. It has been used extensively in both research and commercial environments and has served as the management platform for a number of other ESPRIT and RACE projects in the TMN and distributed systems and service management areas. OSIMIS has been fully in the public domain until version 3.0 to show the potential of OSI management and serve as a benchmark implementation; later versions are still freely available to academic and research institutions for non-commercial use.

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.

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 and Remote Operations Service Elements (ACSE and ROSE) as building blocks for higher level services
- the File Transfer Access and Management (FTAM) and Directory Access Service Element (DASE)
- a ASN.1 compiler with C language bindings (the pepsy tool)
- 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 types of networks

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 object-oriented 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) [X722] language which complements the GMS by producing C++ stub managed objects covering every syntactic aspect and leaving only behaviour to be implemented

• the Remote and Shadow MIB high level object-oriented manager APIs

• a Directory Support service offering application addressing and location transparency services

• 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)

• agents for the OSI version of the Internet TCP/IP MIB (native version) and the OSI transport protocol

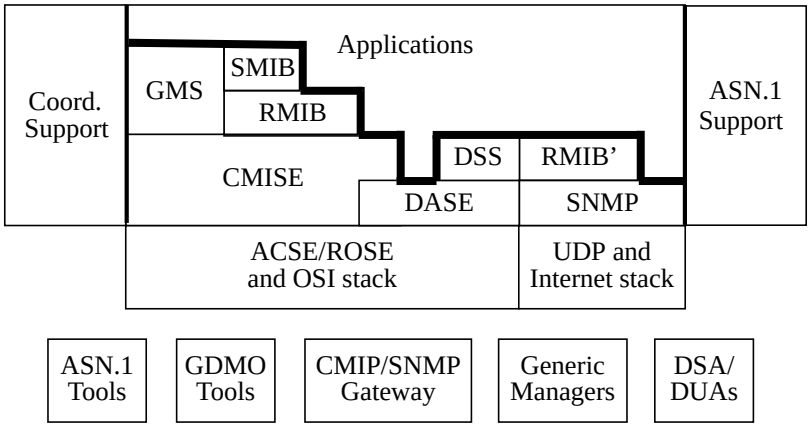

Figure 1 OSIMIS Layered Architecture and Generic Applications**.**

The OSIMIS services and architecture are shown in Figure 1. In the layered part, applications are programs while the rest are building blocks realised as libraries. The lower part shows the generic applications provided; from those the ASN.1 and GDMO tools are essential in providing off-line support for the realisation of new MIBs. The thick line indicates all the APIs an application may use. In practice though most applications use only the Generic Managed System (GMS) and the Remote MIB (RMIB) APIs when acting in agent and manager roles respectively, in addition to

the Coordination and high-level ASN.1 support ones. The latter are used by other components in this layered architecture and are orthogonal to them, as such they are shown aside. Directory access for address resolution and the provision of location transparency may or may not be used, while the Directory Support Service (DSS) API provides more sophisticated searching, discovery and trading facilities.

2 THE ISO DEVELOPMENT ENVIRONMENT

The ISO Development Environment [ISODE] 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 and includes 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) and File Transfer (FTAM) implementations. ISODE is implemented in the C programming language 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 with the ASN.1 support, are used to implement higher level services. In engineering terms, the ISODE stack is a set of libraries linked with applications using it.

ASN.1 manipulation is very important to OSI distributed applications. 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 [XOpen] which, though similar in principle to that of ISODE, is syntactically very different. Translations between the two are possible and such an approach is used to put OSIMIS applications over XOM/XMP.

3 MANAGEMENT PROTOCOL AND O-O 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” call. Associations are represented as communication endpoints (file descriptors) and may be multiplexed to realise event-driven policies.

The OSIMIS CMIS API is known as MSAP (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 implementa-

tions. 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 socket API for Internet UDP access and the ISODE ASN.1 support.

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 possible through thin conversion layers so that the upper layer OSIMIS services are not affected. Regarding ASN.1 manipulation, it is up to an application to encode and decode 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 a 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.

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 used by distributed systems platforms as they are based on RPC which is inherently synchronous with respect to client objects performing remote operations to server objects.

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++ [Pav93b] 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 whose presence is transparent to implementors of specific KSs through the abstract KS interface. This model is depicted in Figure 2.

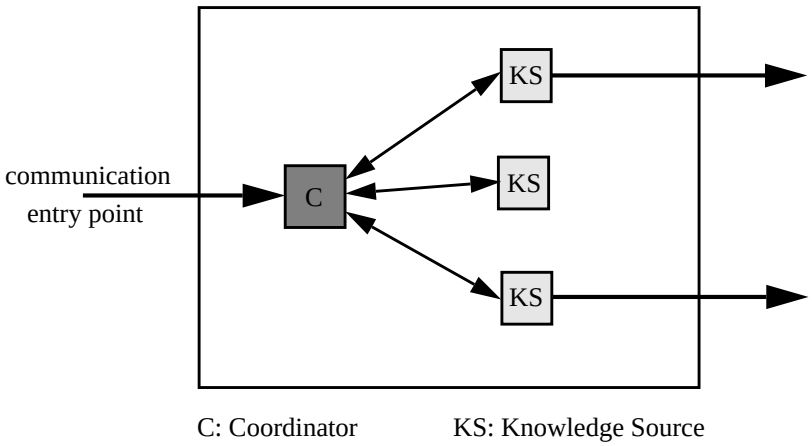

Figure 2 The OSIMIS Process Coordination Support Model.

This coordination mechanism is designed in such a way as to allow integration with other systems' ones. This is achieved through special coordinator derived classes which will interwork with a 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. Such an approach 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 present, the X-Windows Motif and the TCL/TK interpreted scripting language are integrated.

5 THE GENERIC MANAGED SYSTEM

The Generic Managed System (GMS) [Pav93b] [Kni91] 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, Event Reporting and Log Control Systems Management Functions (SMFs), the qualityofServiceAlarm notification of the Alarm Reporting one and partly the Access Control, Metric and Summarization 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 whose instances 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), 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 Fig. 3.

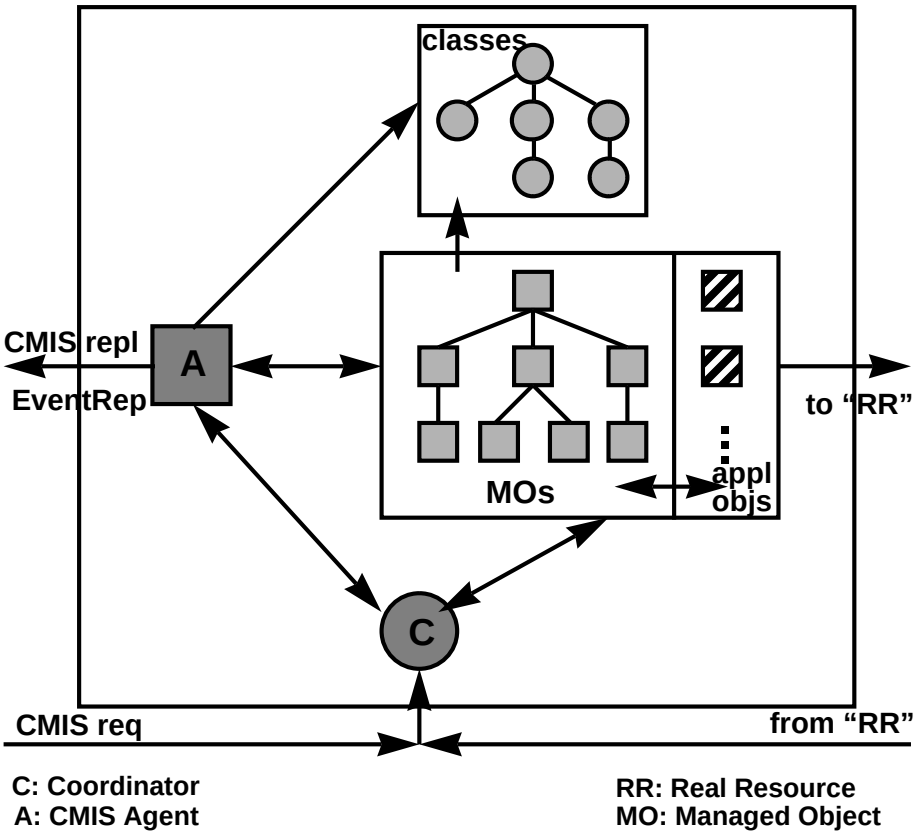

Figure 3 THE GMS Object-Oriented Architecture.

5.1 The CMIS Agent

The CMISAgent is a specialised knowledge source 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 Managed Object Instances and Meta-Classes

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 concerns 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 the “classes” part of Figure 3.

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 - see “MOs” part of Figure 3. Scoping becomes simply a tree search while special care is taken to make sure the tree reflects the state of the associated resources before scoping, filtering and other access operations. 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 to 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 methods 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.

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 and generic attribute and notification types which can be simply instantiated or invoked. This is the case for example with the alarm reporting, metric and summarization 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 fashion transparent to application code. In the case of object management, the code generated 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 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.

5.5 Security

General standards in the area of security for OSI applications are only now being developed while the Objects and Attributes for Access Control Systems Management Function is not yet an International Standard. Nevertheless, systems based on OSI management have security needs and as such OSIMIS provides the following security services:

- peer entity authentication
- data origin authentication and stream integrity
- access control

These were developed in the ESPRIT MIDAS project to cater for the security of management of a large X.400 mail system [Kni94] and will also be used in the RACE ICM project for inter-TMN security requirements on virtual private network applications. Peer entity authentication relies on public key encryption through RSA as in X.509. Data origin authentication is based on cryptographic checksums of CMIP PDUs calculated through the MD5 algorithm. Stream integrity is provided in a novel way that is based on a “well-known” invokeID sequence in ROSE PDUs. It should be noted that as CMIP does not make any provision for the carrying of integrity checksums, these are carried in the ROSE invokeID field. Finally access control is provided through the implementation of the relevant SMF.

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 [Pav94].

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.

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 caching 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. The two models are illustrated in Figure 4.

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.

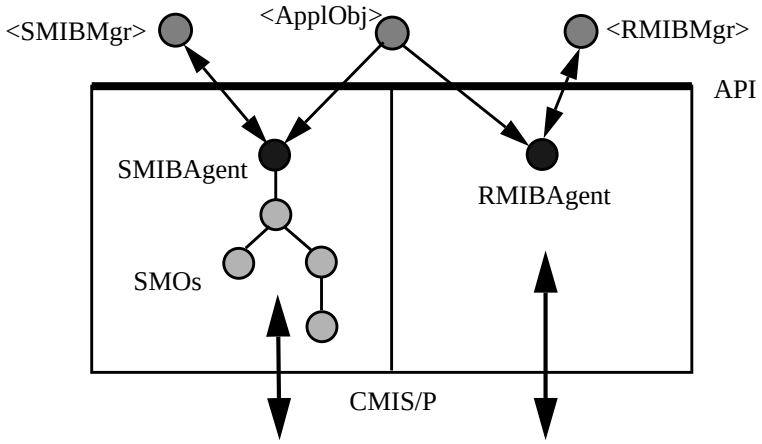

Figure 4 The Remote and Shadow MIB Manager Access Models.

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. Its model structures information in an object-oriented hierarchical fashion similar to that of OSI management. 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 closely resembles 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.

Directory Services can be used 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. In the first level of X.500 usage, the static information residing normally in a local database is converted into directory objects and stored in the directory. 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 where distributed applications 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 management layer (Network Element), as the same application may exist at multiple sites.

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 deregister 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; as such it encompasses libraries providing APIs that can be used to realise applications. Some of these are supported by stand-alone programs such as the ASN.1 and GDMO compilers. Generic management applications are also provided and there are two types of those: semantic-free manager ones that may operate on any MIB without changes and gateways to 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 to perform management operations and provides also a monitoring facility. It is going to be extended with the capability of receiving event reports and of monitoring objects through event reporting. This has recently being re-engineered in TCL/TK.

OSIMIS provides also a set of programs that operate from the command line and realise the full set of CMIS operations. These may 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. The same holds for the relevant information models; the OSI is fully object-oriented while the SNMP supports a simple remote debugging paradigm. Generic application gateways between them are possible without any semantic loss for conversion from CMIS to SNMP as the latter’s operations and information model are a subset of the OSI ones. Work for standards in this area has been driven by the Network Management Forum (NMF) while the RACE ICM project contributed actively to them and also built a generic application gateway based on OSIMIS.

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 extended to cover SNMP version 2. The current approach for the gateway is stateless but the design is such that it allows the easy introduction of stateful optimisations.

9 EPILOGUE

OSIMIS has proved the feasibility of OSI management and especially the 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 provided by most commercial offerings. In complex hierarchical management environments, 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 easy construction of proxy systems. 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 object-oriented distributed systems concepts and the protocol-based management world can coexist by combining the OSI Directory (X.500) and Management (X.700) models.

OSIMIS projects a management architecture in which OSI management is used as the unifying technology which integrates other technologies through application level gateways. The OSI management richness and expressive power guarantees no semantic loss, at least with respect to SNMP or other proprietary technologies. The emerging of the OMG CORBA distributed object-oriented framework is expected to challenge OSI management in general and platforms such as OSIMIS but there is potential for harmonious coexistence. Research work is envisaged in supporting gateways to CORBA systems and vice-versa, OSI management-based systems over CORBA, lightweight approaches to avoid the burden and size of OSI stacks through service relays, interpreted policy languages, management domains, sophisticated discovery facilities etc.

Acknowledgements

Many people have contributed to OSIMIS to be mentioned in this short space here. James Cowan of UCL though should be mentioned for the innovative design and implementation of the platform independent GDMO compiler, Thurain Tin, also of UCL, for the excellent RMIB infrastructure and Jim Reilly of VTT, Finland for the SNMP to GDMO information model translator that was produced over a week-end(!) and the first version of the metric objects. This work was carried out under the RACE ICM and NEMESYS and the ESPRIT MIDAS and PROOF projects.

10 REFERENCES

[Strou] Stroustrup B., The C++ Programming Language, Addison-Wesley, Reading, MA, 1986

[X701] ITU X.701, Information Technology - Open Systems Interconnection - Systems Management Overview, 7/91

[X722] ITU X.722, Information Technology - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects, 8/91

[SNMP] Case J., M. Fedor, M. Schoffstall, J. Davin, A Simple Network Management Protocol (SNMP), RFC1157, 5/90

[Pav93a] Pavlou G., S. Bhatti and G. Knight, Automating the OSI to Internet Management Conversion Using an Object-Oriented Platform, IFIP Conference on LAN/MAN Management, Paris, 04/93

[OMG] Object Management Group, The Common Object Request Broker: Architecture and Specification, Document Number 91.12.1, Revision 1.1, 12/91

[ISODE] Rose M.T., J.P. Onions, C.J.Robbins, The ISO Development Environment User's Manual version 7.0, PSI Inc. / X-Tel Services Ltd., 7/91

[XOpen] X/Open, OSI-Abstract-Data Manipulation and Management Protocols Specification, 1/92

[Pav93b] Pavlou G., Implementing OSI Management, Tutorial Presented at the 3rd IFIP/IEEE ISINM, San Francisco, 4/93, UCL Research Note 94/74

[Kni91] Knight G., G. Pavlou, S. Walton, Experience of Implementing OSI Management Facilities, Integrated Network Management II, ed. I. Krishnan / W. Zimmer, pp. 259-270, North Holland, 1991

[Kni94] Knight G., S. Bhatti, L. Deri, Secure Remote Management in the ESPRIT MIDAS Project, IFIP Upper Layer Protocols, Architectures and Applications conference, Barcelona, 5/94

[Pav94] Pavlou G., T. Tin, A. Carr, High-Level Access APIs in the OSIMIS TMN Platform: Harnessing and Hiding, Towards a Pan-European Telecommunication Service Infrastructure, ed. H.J. Kugler, A. Mullery, N. Niebert, pp. 181-191, Springer Verlag, 1994

[X500] ITU X.722, Information Processing - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Service, 1988

11 BIOGRAPHIES

George Pavlou received his Diploma in Electrical, Mechanical and Production Engineering from the National Technical University of Athens in 1982 and his MSc in Computer Science from University College London in 1986. He has since worked in the Computer Science department at UCL mainly as a researcher but also as a teacher. He is now a Senior Research Fellow and has been leading research efforts in the area of management for broadband networks, services and applications.

Kevin McCarthy received his B.Sc. in Mathematics and Computer Science from the University of Kent at Canterbury in 1986 and his M.Sc. in Data Communications, Networks and Distributed Systems from University College London in 1992. Since October 1992 he has been a member of the Research Staff in the Department of Computer Science, involved in research projects in the area of Directory Services and Broadband Network/Service Management.

Saleem N. Bhatti received his B.Eng.(Hons) in Electronic and Electrical Engineering in 1990 and his M.Sc. in Data Communication Networks and Distributed Systems in 1991, both from University College London. Since October 1991 he has been a member of the Research Staff in the Department of Computer Science, involved in various communications related projects. He has worked particularly on Network and Distributed Systems management.

Graham Knight graduated in Mathematics from the University of Southampton in 1969 and received his MSc in Computer Science from University College London in 1980. He has since worked in the Computer Science department at UCL as a researcher and teacher. He is now a Senior Lecturer and has led a number of research efforts in the department. These have been concerned mainly with two areas; network management and ISDN.