This PDF file contains a chapter of:

# INTEGRATED COMMUNICATIONS MANAGEMENT OF BROADBAND NETWORKS

*Crete University Press, Heraklio, Greece*
*ISBN 960 524 006 8*

*Edited by David Griffin*



First published in 1996 by
CRETE UNIVERSITY PRESS
Foundation for Research and Technology
P.O. Box 1527, Heraklio, Crete, Greece 711 10
Tel: +30 81 394235, Fax: +30 81 394236
email: pek@iesl.forth.gr



The ICM consortium consists of the following companies:

Alcatel ISR, France
Alpha SAI, Greece
Ascom Monetel, France
Ascom Tech, Switzerland
Centro de Estudos de Telecommunicações, Portugal
Cray Communications Ltd., United Kingdom (Prime contractor)
Danish Electronics, Light & Acoustics, Denmark
De Nouvelles Architectures pour les Communications, France
Foundation for Research and Technology - Hellas, Institute of Computer Science, Greece
GN Nettest AS, Denmark
National Technical University of Athens, Greece
Nokia Corporation, Finland
Queen Mary and Westfield College, United Kingdom
Unipro Ltd., United Kingdom
University College London, United Kingdom
University of Durham, United Kingdom
VTT - Technical Research Centre of Finland

# *Chapter 10*

## The OSIMIS TMN platform

Editors: Kevin McCarthy, George Pavlou
Authors: George Pavlou, Kevin McCarthy, Thurain Tin, James Cowan,
George Mykoniatis, Jorge Sanchez, James Reilly,
Costas Stathopoulos, Stelios Sartzetakis, David Griffin,
Jose Neuman de Souza, Nazim Agoulmine,
Henryka Jormakka, Juha Koivisto

The OSIMIS (OSI Management Information Service) platform provides the foundation for the quick and efficient construction of complex TMN systems. It is an object-oriented development environment in C++ [10.1], based on the OSI Management Model [10.5], which hides the underlying protocol complexity (CMIS/CMIP [10.6][10.7]) and harnesses the power and expressiveness of the associated information model [10.8]. OSIMIS combines the thoroughness of the OSI models and protocols with distributed systems concepts projected by ODP [10.19] to provide a highly dynamic distributed information store. It also combines seamlessly the OSI management power with the large installed base of Internet SNMP-capable network elements.

OSIMIS is ideally suited for Telecommunications Management Network (TMN) [10.3] 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/P to SNMP application gateway [10.34], whilst adaptation to other models or proprietary systems is feasible. OSIMIS projects a model whereby OSI management, being the most powerful of current management technologies, provides the unifying end-to-end means through which

other technologies are integrated via application level gateways, possibly in a generic fashion. This chapter explains the OSIMIS concepts, architecture, components and philosophy.

Throughout this chapter, the term *management application* (or simply *application*) refers to a TMN physical block e.g. Operations System (OS), Workstation-Operation System (WS-OS), Mediation Device (MD), Q-Adaptor (QA) or Network Element (NE). All these blocks, apart from WS-OSs, export *managed objects* across management interfaces. Also, all these blocks, apart from NEs, may contain managing objects. The term *object* throughout this chapter refers to a managed, managing or support infrastructure object (realised as a C++ instance in engineering terms).

## 10.1 Introduction

OSIMIS is an object-oriented management platform based on the OSI model [10.5] and implemented mainly in C++. 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 or an object within an application can be in both managing and managed roles. This is particularly common in situations where a management system is decomposed according to a hierarchical logically layered approach. This is exactly the model suggested by the TMN and OSIMIS provides special support to realise management hierarchies.

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 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 [10.27] and the emerging OMG CORBA [10.21] technologies. OSIMIS already provides a generic application gateway between CMIS [10.6] and SNMP [10.27] while a similar approach for integrating OSI management and the OMG CORBA framework may be possible.

OSIMIS has been developed in a number of European research projects, in addition to ICM, namely the RACE NEMESYS and ESPRIT MIDAS and PROOF projects. It has been used extensively in both research and commercial environments and has served as the management platform for a number of other RACE and ESPRIT projects in the TMN and distributed systems management areas.

### 10.1.1 What is meant by “TMN platform?”

OSIMIS is principally an object-oriented development environment: it is a distributed systems management toolkit that supports the easy construction of TMN components such as Operations Systems, Q-Adaptors, Mediation Devices and Network Element agents. Development is greatly assisted by the provision of generic support infrastructure such as GDMO/ASN.1 compilers, run-time libraries at various levels of abstraction, directory systems and generic manager applications such as MIB instance browser.

Being a distributed management platform, OSIMIS provides an implementation of the CMIP protocol together with a fully integrated X.500-based location transparency mechanism in order to meet the communication and distribution needs of TMN applications. Lightweight security mechanisms are also provided to maintain the access rights, integrity and confidentiality of relevant associations.

### 10.1.2 Platform component overview

OSIMIS uses the ISODE (ISO Development Environment) [10.29] as the underlying OSI upper layer protocol stack but also supports XOM/XMP [10.22] as an alternative. The advantages of the ISODE environment include the provision of services like FTAM [10.20] and a full implementation of the OSI Directory Service (X.500) [10.4] which are essential in complex distributed 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, whilst interoperation of applications between any of these is possible via Transport Service Bridging.

The OSIMIS services and architecture are shown in Figure 10.1. 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 APIs. 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.

The OSIMIS platform comprises the following types of support:

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

Some of these services are provided by ISODE (see Figure 10.2) and these are:

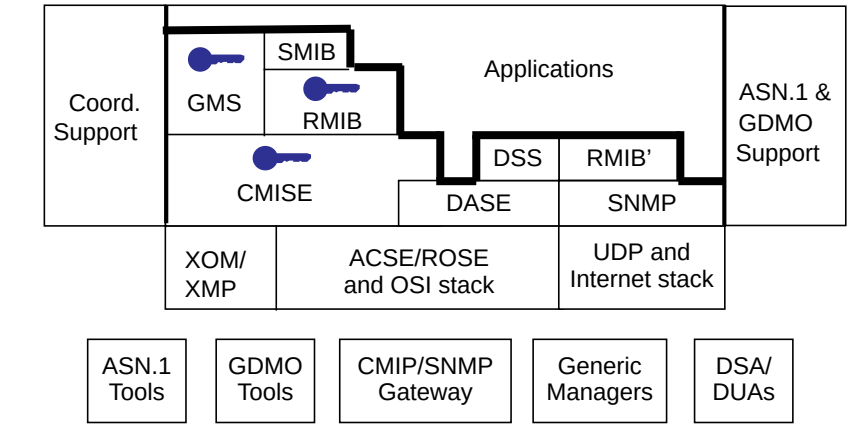



**Figure 10.1 OSIMIS layered architecture and generic applications**

- 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),
- an ASN.1 compiler with C language bindings (the pepsy tool),
- a Remote Operations stub generator (the rosy tool),
- an 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,
- high-level ASN.1 support that encapsulates ASN.1 syntaxes in C++ objects,
- an ASN.1 Attribute compiler which uses the ISODE pepsy compiler to automate to a large extent the generation of syntax C++ objects,
- a Coordination mechanism that allows an application to be structured in a fully event-driven fashion and can 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

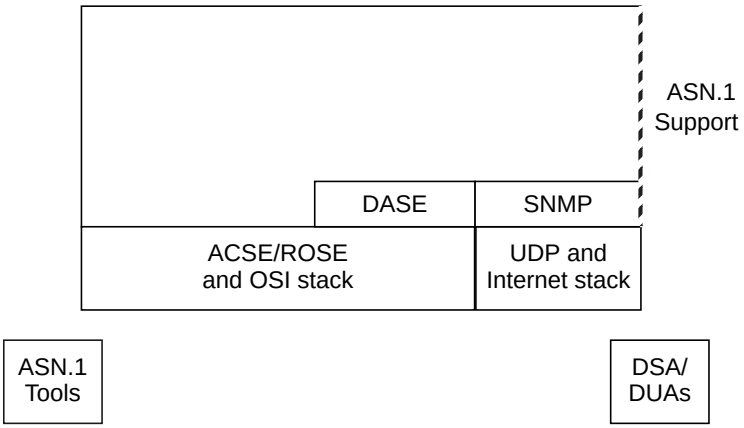



**Figure 10.2 The components provided by ISODE**

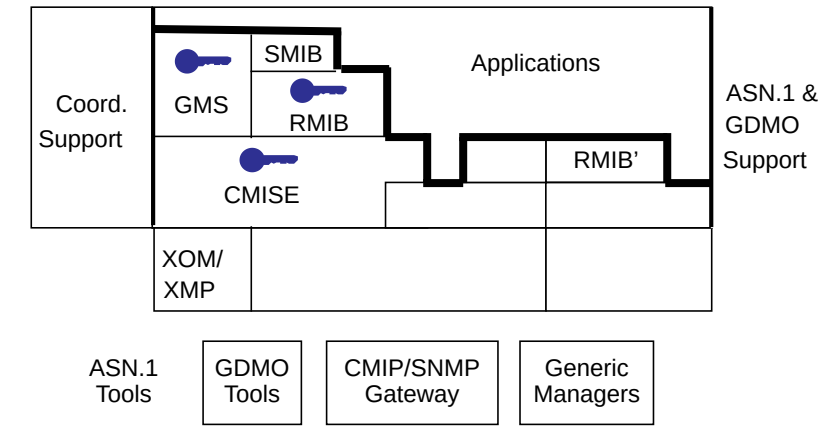



**Figure 10.3 The components provided by OSIMIS**

library of generic attributes, notifications and objects and systems management functions,

- a suite of lightweight security mechanisms, which meet many requirements for access control, authentication, data integrity and data confidentiality,
- a compiler for the OSI Guidelines for the Definition of Managed Objects (GDMO) [10.10] 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/P to SNMP application gateway driven by a translator between SNMP and OSI GDMO MIBs,
- a set of generic manager applications.

## 10.1.3 Underlying communications environments

### 10.1.3.1 The ISO development environment

The ISO Development Environment (ISODE) [10.29] 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/ITU-T recommendations and includes tools for ASN.1 manipulation and remote operations stub generation. Two fundamental OSI applications are provided: Directory Service (X.500) [10.4] and File Transfer (FTAM) [10.20] implementations. ISODE is implemented in the C programming language [10.2] and runs on most versions of the UNIX operating system. ISODE does not provide any network and lower layer protocols e.g. X.25 or 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

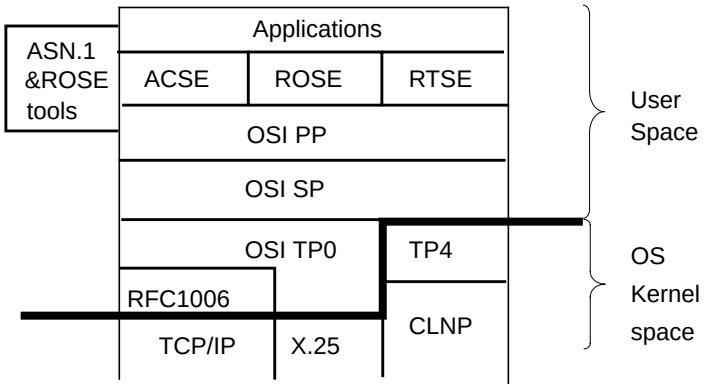



**Figure 10.4 The ISODE OSI stack**

protocols of the OSI seven-layer model. The Association Control, Remote Operations and Reliable Transfer application layer Service Elements (ACSE, ROSE and RTSE) are also provided which, when used in conjunction with the ASN.1 support, act as building blocks for 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) stack. 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 on 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 between PEs and the encoded data stream according to the encoding rules (e.g. BER).

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) [10.30]. 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 [10.26], 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 sub-networks 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.

#### 10.1.3.2 Industry standard XOM/XMP communication environment

The X/Open Management Protocols (XMP) [10.22] defines an Application Program Interface (API) that provides access to management information services. The services are defined in terms of operations and notifications on managed objects. The interface uses the generic concepts defined by ISO, which form the basis for systems management, and supports the model defined in the Structure of Management Information (SMI). It offers service primitives which correspond to the abstract services of both CMIS and SNMP, and intends to keep management software independent of the implementation of the protocol stack.

The interface is designed to be used and implemented in conjunction with the general-purpose OSI-Abstract-Data Manipulation (XOM) API [10.22]. OSI data corresponds to information structured by the Abstract Syntax Notation One (ASN.1). Data manipulation consists of creating, examining, modifying and deleting data corresponding to ASN.1 syntax. By providing tools for manipulating ASN.1 objects, the interface shields the programmer from much of ASN.1's complexity.

While XOM/XMP takes an object-oriented view of structural information, it does not incorporate all the characteristics of other object-oriented systems. In particular, the implementations of the functions for manipulating objects are separate from the definitions of the objects' classes, and there is no notion of encapsulating or hiding the information associated with objects (although the interface does hide the information representation).

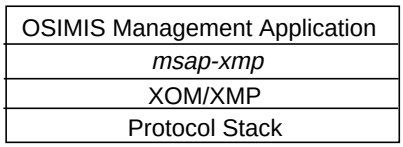



**Figure 10.5 OSIMIS XOM/XMP application stack**

The initial implementation of high-level OSIMIS services required an underlying ISODE OSI stack. The ISODE ASN.1 and OSIMIS CMIS APIs are different from the current industry standard XOM/XMP as they were designed much earlier. To make it possible to use protocol stacks other than that provided by ISODE, an XMP support library *msap-xmp* (see Figure 10.5) was developed in ICM. This library provides a link between the XOM/XMP API and OSIMIS by replacing the original CMIP library used in the ISODE version of OSIMIS, whilst still retaining the OSIMIS CMIS API, known as Management Service Access Point (MSAP) - see Section 10.2.1, so that the OSIMIS management applications may operate over either an XMP or an ISODE stack.

## 10.2 The OSIMIS components and services

### 10.2.1 Management protocol and 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 service elements together with their 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 end-points (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 long 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 10.2.4. OSIMIS also includes an implementation of the Internet SNMPv1 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, again without conforming to XMP. Applications using CMIS need to manipulate ASN.1 types for the CMIS managed object attribute values, actions, 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 (see Section 10.1.3.2). Such conversion infrastructure was first designed in the ESPRIT MIDAS project while it has been implemented in the RACE ICM project.

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 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. This high-level OO ASN.1 approach is used by higher level OSIMIS services such as GMS, RMIB and SMIB.

### 10.2.2 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 a number of different techniques to organise an application for handling both external and internal events. The organisation needs to be event driven so that no resources are used when the system is idle. The two major techniques are:

- use a single-threaded execution paradigm, and,
- use a multi-threaded one.

In the first, external communications should follow an asynchronous model, since waiting for the 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, this requirement is fulfilled by the OSIMIS Application Coordination Support. In the second, many threads of control can execute 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.

The advantage of the first mechanism is that it is supported by most operating systems and, as such, it is lightweight and efficient whilst 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 operating system mechanism.

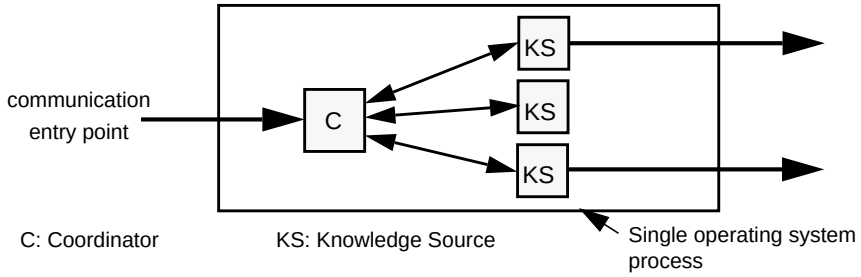



**Figure 10.6 The OSIMIS process coordination support model**

OSIMIS provides an object-oriented infrastructure in C++ which allows an application to be organised in a fully event-driven fashion under 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, whilst the KS is an abstract class that facilitates usage of the coordinator services and integrates external sources of input and 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 10.6.

This coordination mechanism is designed in such a way as to allow integration with those provided by other systems; which is achieved through special classes derived from the coordinator, allowing interworking with a particular mechanism. These specialised coordinator classes still control the sources of input and timer alarms of the OSIMIS KSs, but pass on the task of performing the central listening to the other system's coordination mechanism. This is required 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. A similar mechanism is needed for Graphical User Interface technologies which have their own coordination mechanisms: In which case, a new specialised coordinator class is needed for each of them. At the present time, the X-Windows Motif, the Tcl/Tk interpreted language and the InterViews graphical object library are fully integrated.

## 10.2.3 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 fully supports the *Object Management* [10.11], *Event Reporting* [10.13] and *Log Control* [10.14] Systems Management Functions (SMFs); the qualityofServiceAlarm notification of the *Alarm Reporting* [10.12] SMF; together with profiles of the *Access Control* [10.17], *Monitor Metric* [10.15] and *Summarisation* [10.16] SMFs. In conjunction with the GDMO

compiler the GMS 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 also contains generic attribute types such as counter, gauge, counterThreshold, gaugeThreshold and tideMark, and specific attributes and objects as in the Definition of Management Information (DMI) [10.9], 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 10.7. It should be noted that this is one engineering "capsule" in ODP terms i.e. one operating system process.

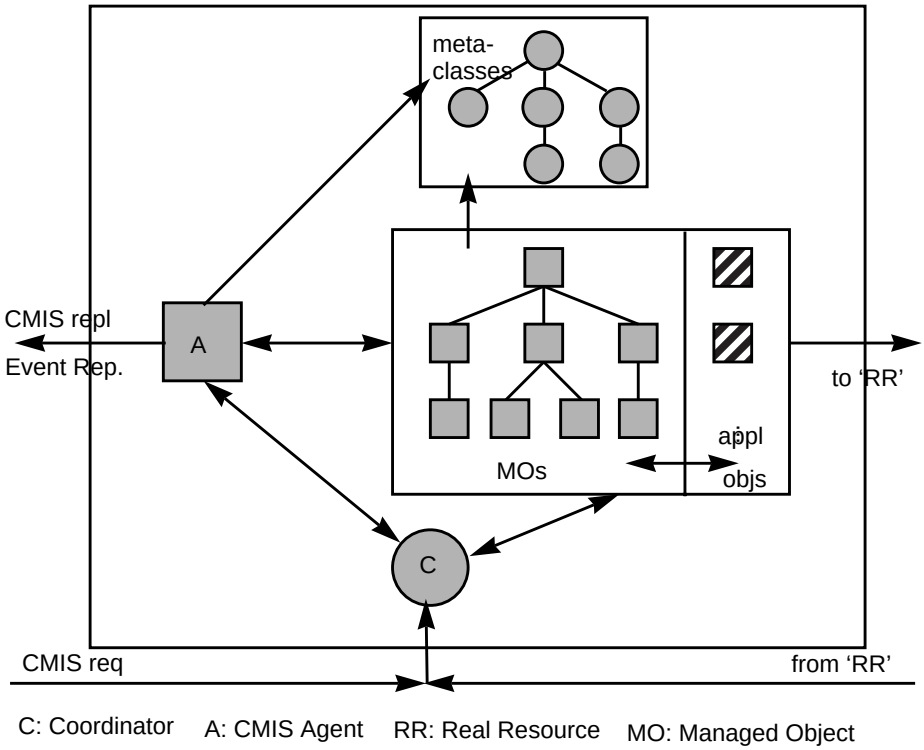



**Figure 10.7 The GMS object-oriented architecture**

#### 10.2.3.1 The CMIS agent

The CMISAgent is a specialised knowledge source, more specifically a specific Isode-Agent, as it accepts management associations. There is always one instance of this class for every application undertaking an agent role. Its functions are to accept or reject 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, which is at present purely synchronous: a method call should always return with a result i.e. attribute values or an 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 manager.

#### 10.2.3.2 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 related 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 'meta-classes' block of Figure 10.7.

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 the "MOs" part of Figure 10.7. Scoping becomes simply a tree search, whilst 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.)

#### 10.2.3.3 Real resource access

There are three possible types of interaction between the managed object and the associated resource[1] with respect to CMIS Get requests:

- access upon external request,
- 'cache-ahead' through periodic polling of the real resource,
- update through asynchronous event reports from the real resource.

The first one means that no real resource access is performed until a managing process accesses the managed object. 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 methods through the coordination mechanism. When asynchronous reports or results are required, it is likely that a separate object will be needed to demultiplex the incoming information and deliver it to the appropriate managed object instance. It should be 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 as Internal Communications Controllers (ICCs) and are essentially specialised knowledge source objects.

The internal organisation of a GMS-based agent in terms of the major interacting object instances is shown in Figure 10.7. 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 number of other objects which are application specific. Note also that the OSI stack is essentially encapsulated by the CMISAgent object.

#### 10.2.3.4 Systems management functions: overview

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 summarisation objects. In other cases, the GMS knows the semantics of these classes and uses them accordingly e.g. in access control, 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 the object management SMF the code generated by the GDMO compiler together with the GMS completely hiding the emission of object creation and deletion notifications, as well as 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 val-

1. In the TMN context 'real resource' may refer to resources at high levels of abstraction, mapped onto lower level resources as well as to the finest granularity resources at the lowest level of the management hierarchy i.e. network elements.

ues. 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.

#### 10.2.3.5 Monitoring and summarisation SMFs

While the manager-agent model does not in itself restrict the amount of intelligence that may be incorporated and provided by managed objects, most standard specifications concentrate on providing a rich enough set of attributes and actions which model information and possible interactions with the underlying resource. Notifications are also provided to report significant exceptions but they are usually generic ones, e.g. object creation, object deletion and attribute value change.

The OSI Structure and Definition of Management Information (SMI/DMI) specify generic attribute types such as counter, gauge, threshold and tide-mark. Gauges model entities with associated semantics e.g. number of calls, users, quality of service etc. or the rate of change of associated counters e.g. bytes per second. Thresholds and tide-marks may be applied to gauges and generate QoS alarms and also attribute value changes, indicating change of the high or low 'water mark'. Such activities are of a *managing* nature. Although thresholding functions could be made part of managed objects modelling real resources, it does not take long to recognise their genericity. As such, they are better provided elsewhere so that they become re-usable. The relevant ISO/ITU-T group recognised the importance of generic monitoring facilities and standardised the Metric Monitor [10.15] and Summarisation [10.16] systems management functions. By making such functions generic, it is possible to implement them once and make them part of the associated platform infrastructure.

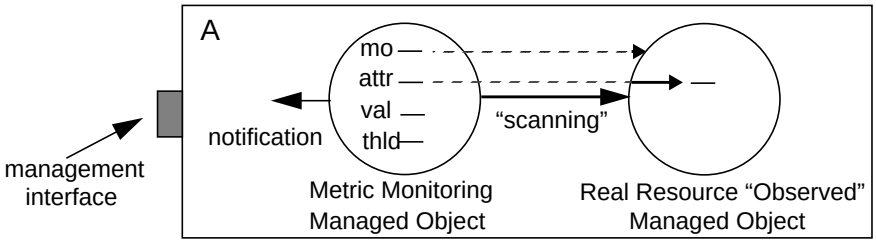
The whole idea behind monitor metric objects is to provide thresholding facilities in a *generic* fashion. Monitor metric objects may be instantiated within an application in an agent role and be configured to monitor, at periodic intervals, an attribute of another real resource managed object. That attribute may belong to a network element managed object but also to a higher-level management application, being mapped onto lower-level managed objects through an Information Conversion Function (ICF). The observed attribute should be a counter or gauge and the metric object either observes it 'as is' or converts the observed values to a rate (derived gauge) over time. Statistical smoothing of the observed values is also possible if desired.

The main importance of this facility is the attachment of gauge thresholds and tide-marks to the resulting derived gauge which may generate quality of service alarms and indicate the high and/or low 'water mark', as desired by systems using this function. In fact, the metric objects essentially enhance the 'raw' information model of the observed object. The metric monitor functionality can be summarised as:

- *data capture:* through observation or 'scanning' of a managed object attribute,
- *data conversion*: potential conversion of a counter or gauge to a derived gauge,
- *data enhancement:* potential statistical smoothing of the derived result,
- *notification generation:* QoS alarm and attribute value change notifications.

The metric monitoring model is shown in Figure 10.8.

Gauge thresholds are always specified in pairs of values: a triggering and a cancelling threshold. The former will generate a notification when crossed only if the latter




A Application in agent role ("exporting" a management interface)
— Attribute

**Figure 10.8 The metric monitoring model**

has been previously crossed in the opposite direction. This prevents the continuous generation of notifications when the measured value oscillates around the triggering threshold and is known as the *hysteresis* mechanism. Figure 10.9 shows both high (e.g. 'over-utilisation') and low thresholds (e.g. 'under-utilisation') applied to the observed value. Typically, a normalised value of around 5% is used as the hysteresis interval.

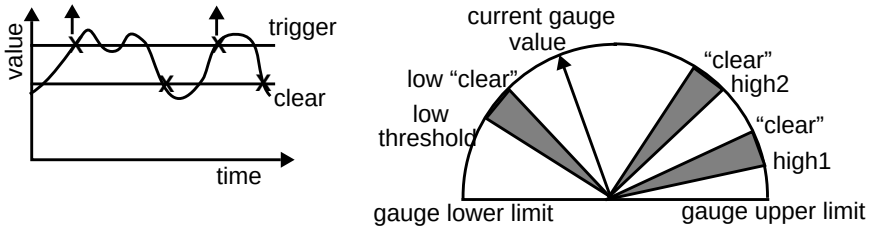



**Figure 10.9 Thresholding with hysteresis**

The metric objects offer the OSI management power through event reporting and logging, even if the 'raw' observed management information model does not support such notifications. More importantly, they obviate the use of rates, thresholds and tide-marks in a way tied to specific managed object classes but they allow the same flexibility and power dynamically, whenever a managing system needs it. Such a monitoring facility reduces the management traffic between applications and their impact on the managed network by supporting an event-based operation paradigm.

The summarisation objects extend the idea of monitoring a single attribute to monitoring many attributes across a number of selected managed object instances. They offer similar but complementary facilities to metric objects. In this case, there are no comparisons or thresholding but only the potential statistical smoothing and simple algorithmic results of the observed values (min, max, mean and variance). These are reported periodically to the interested managing systems through notifications. The observed managed objects and attributes can be specified either by supplying explicitly their names or through CMIS scoping and filtering. The observed values may be raw ones, modelling an underlying resource, or enhanced values as observed by metric objects. They can be reported either at every observation period or after a number of observation periods (buffered scanning).

The major importance of this facility is that a number of measurements a managing system needs can be specified once and then reported as mentioned, without the managing system having to send complex CMIS queries periodically or having to perform the statistical smoothing etc. Intelligence in this context has to do with the periodic scanning and reporting of diverse information automatically, after the criteria for its assembly have been specified once. Such criteria may be expressed through CMIS scoping and filtering, providing a lot of flexibility in summarising information of dynamic nature (e.g. traffic across certain ATM virtual path connections etc.) Any other complex computations on the summarised information are left to be performed by the managing system. Such logic is usually of static, pre-compiled nature.

#### 10.2.3.6 Advanced intelligent monitoring facilities

While the metric monitor and summarisation facilities can be combined to provide a lot of flexibility, they still leave the task of complex calculations and comparisons to managing systems. In most cases it is not just a single attribute value that needs to be monitored and compared to thresholds but the combination of more than one attribute, possibly across different managed objects. The derived value is thus the result of the application of a *mathematical operation* on the observed values. Such an operation could be the $sum(a_1+...+a_n)$, *div(a/b)*, *diff(a-b)* etc. The combination of more than one operations may be used to construct arbitrary expressions, albeit at the cost of multiple intelligent monitoring objects.

A facility therefore is needed that combines the properties of the metric monitor (thresholding on a derived result) and summarisation objects (observation of arbitrary objects and attributes). This combination is effected through a mathematical operation that is performed on the observed values to yield the derived result. We have recognised the need for such a facility early on and have specified and implemented a number of GDMO classes that provide this functionality, termed collectively *Generic Support Monitoring Objects* [10.38]. Since then, the relevant ITU-T standardisation committee has also recognised this need and provided amendments to the metric and summarisation functions offering similar functionality.

In most cases, simple mathematical calculations are enough to express real-world problems. For example, if connection acceptance and rejection counts are kept, the connection rejection ratio is given by the simple formula below. Such arithmetic operations can be either specified statically, through a relevant attribute of the summarisation object, or specified dynamically by combining the observed attribute values with operands. In the latter case, ultimate flexibility is provided in applying arbitrary mathematical operations.

$$\text{Connection Rejection Ratio} = \frac{\text{Sum(connRej)}}{\text{Sum(connAcc)} + \text{Sum(connRej)}} \qquad \text{Transmission error rate} = \frac{\text{Sum(pdusRej)}}{\text{Sum(pdusSent)}}$$

##### *10.2.3.6.1 Model*

The underlying concepts for the Generic Monitoring Support function are based on the combination of the Metric Monitoring and Summarisation functions. This function provides the ability to aggregate attribute values, apply operations on them and provide

statistical information about the aggregated result of the operation. The function is realised by the Generic Support Monitoring Objects (GSMO) and provides for:

- the ability to monitor information provided by single or multiple attribute types,
- the selection of the observed attribute values either explicitly, through the names of the containing objects, or through scoping and filtering,
- the identification of an operation (e.g. sum) to be applied on the observed attribute values,
- the identification of an algorithm (e.g. uniform weighted moving average) used to smooth the derived result,
- the ability to emit notifications when the derived result crosses a threshold or "pushes" a tide-mark or when any of its attributes change.

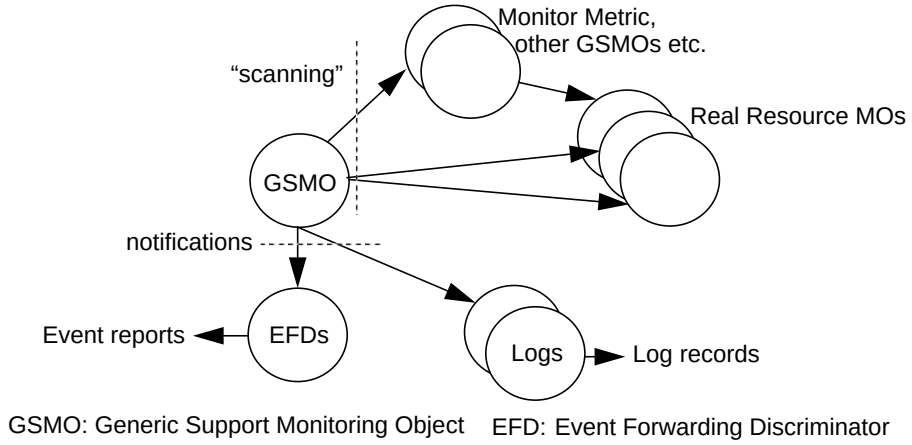



**Figure 10.10 Intelligent monitoring and summarisation model**

The model for the operation of the GSMO objects is similar to that of summarisation and is shown in Figure 10.10. Information is obtained by observing attributes of other objects, including managed objects representing a management view of an underlying network or service resource, metric objects, or other GSMO objects. The attributes of those "observable" objects are accessed at the object boundary, triggering associated behaviour in the same fashion as across a management interface.

## 10.2.4 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 whilst work in this area was taken much further in ICM.

### 10.2.4.1 The remote MIB (RMIB) and the shadow MIB (SMIB)

The Remote MIB (RMIB) support service offers a higher level API which provides the abstraction of a proxy agent 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 proxy 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 all be 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, or a timeout occurs, whilst this is not the case with the asynchronous API. The introduction of threads or co-routines will obviate the use of the asynchronous API for reasons other than event reporting or a one-by-one delivery mechanism for linked replies.

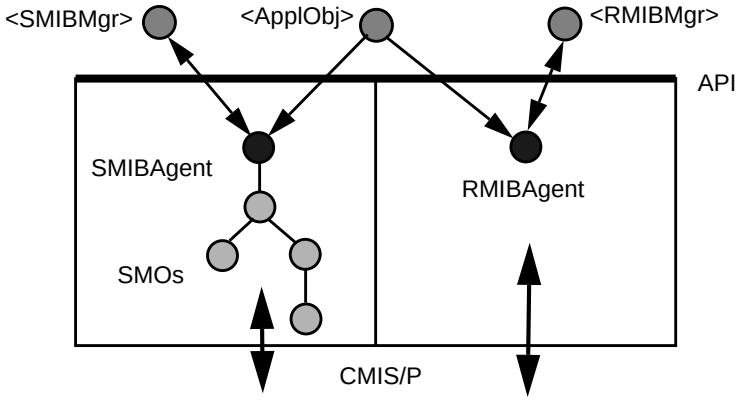



**Figure 10.11 The remote and shadow MIB manager access models**

While the RMIB infrastructure offers a much higher level facility than a raw CMIS API such as the OSIMIS MSAP or X/Open's XMP. Its message passing 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 two fold: first, the API could be less CMIS-like for accessing the local objects since parameters such as distinguished names, scoping etc. can be replaced by pointers in the 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. The model and supporting C++ classes are very similar to the RMIB ones. The two models are illustrated in Figure 10.11.

Both the RMIB and SMIB support services are based on a compiled model, whilst 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, which were used in ICM 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 front ends.

#### 10.2.4.2 The Tcl-RMIB API

Being both interpreted and string based, the Tcl language was selected to form the basis for the OSIMIS scripting language *Tcl-RMIB*. The presence of existing string based interfaces to CMIS, via the RMIB, has greatly assisted in the development of this new interface.

Tcl-RMIB [10.33] assists rapid prototyping of management GUIs, event monitors, MIB browsers etc. with similar functionality to that of the RMIB API, supporting both synchronous and asynchronous interactions as shown in Figure 10.12.

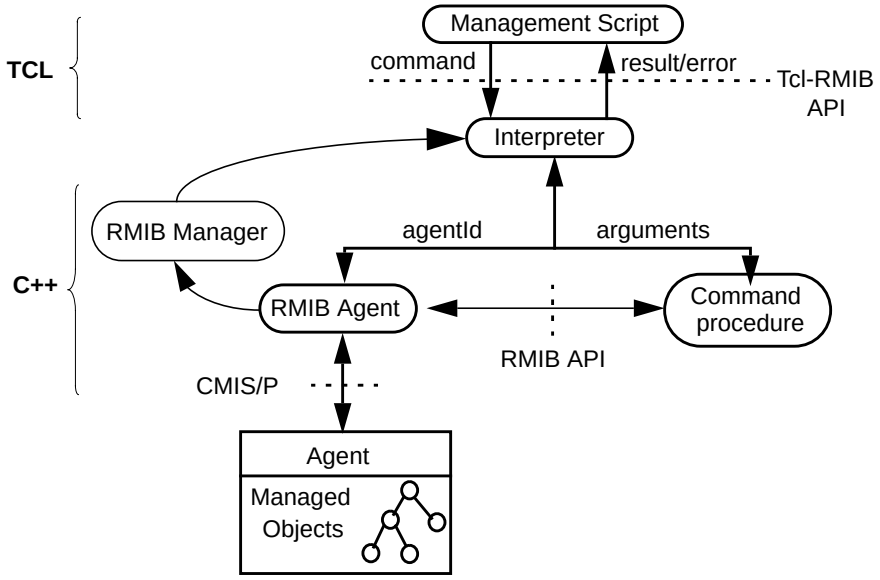



**Figure 10.12 Tcl-RMIB model interactions**

## 10.2.5 Security services

### 10.2.5.1 Introducing secure management services

General standards in the area of security for OSI applications are only now being developed, whilst the *Objects and Attributes for Access Control* Systems Management Function [10.17] has only recently become an international standard. Nevertheless, systems based on OSI management have security requirements and as such OSIMIS provides the following security services: peer authentication, stream integrity, data confidentiality and access control.

If we consider the scenario of a TMN process that is connected to another component via the X-interface by a network link, then there are a number of threats posed by this link:

- Passive attack: this allows an interloper to *observe data* passing on the link.
- Active attack: where a process is interposed on the link in order to *modify* or *inject data* on the link. There are four important sub-classes:
  - The *replay* of a previous message, e.g. one that causes a CMIS Action to be re-applied that leads to a managed element resource being reset.
  - The *modification* of a previous message may result in an unauthorised effect, e.g. the application of a CMIS Set operation to an administrative state attribute.
  - *PDU reordering, insertion or removal.*
  - The *masquerade* attack involves an entity attempting to take on the identity of another entity that has important privileges. This might be achieved by the modification of a previously captured message.

The security of network management is probably the ultimate example of the need to trade off the level of security achieved against any reduction in the timeliness that operations must be enacted: To this end OSIMIS enlists the services of lightweight *Data Encryption Standard (DES)* private key mechanisms to defeat the attacks listed above. Private key mechanisms require that both parties (i.e. TMN processes) that need to set up an association, have knowledge of the same *key.* The key is a 64 bit string that is used as the input parameter to the DES algorithm; which causes a defined translation of the input data to the output data stream, in such a manner that the input data can only be re-generated from the output data stream if the original key is known.

### 10.2.5.2 Lightweight authentication, data integrity and confidentiality

The mechanisms are lightweight in the sense that they are based on an optimised *Data Encryption Standard* (DES) [10.35] implementation; as opposed to the significantly more expensive *Rivest, Shamir and Adleman* (RSA) [10.36] cryptographic algorithm, with its associated overhead of X.500 certificate storage.

#### *10.2.5.2.1 DES secret key authentication*

This assumes that only the two processes involved in an association establishment have knowledge of the association initiator's secret key. The authenticated association set up involves the exchange of a shared 'secret' value (the *session key*), that will be gener-

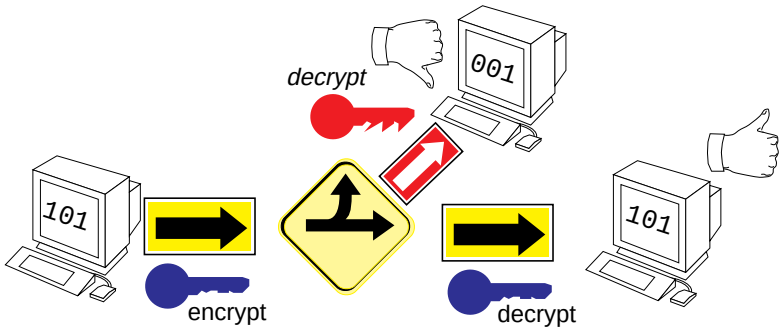



**Figure 10.13 Successful DES decryption requires knowledge of the DES encryption key**

ated on a per association basis. A time value is included in the set up so as to ensure that a partially completed set up message exchange can not be replayed by a rogue process.

Successful completion of the authenticated association set up ensures that the identities of both parties (as denoted by globally unique Distinguished Names) is as claimed. This prevents a rogue managing application from gaining illegal access to and/or control of an agent process; or a rogue agent process imitating a valid agent process and supplying incorrect information to a managing application.

#### *10.2.5.2.2 Data integrity and PDU ordering guarantees*

Data integrity and PDU ordering are achieved through the usage of *Message Digest 5 (MD5)* checksums and PDU sequence numbers respectively. Each CMIP PDU is first encoded using the Distinguished Encoding Rules (DER) to yield a byte stream `B`. The session key `Ks` (which was exchanged during association set up) is then appended to yield the byte stream `Bs.` Next `Bs` is used as input to the MD5 hashing algorithm which yields a 128 bit checksum (`c`) which is carried in the ROS `invokeId` field.

```
Bs = B*Ks, c = MD5hash(Bs)
```

We can guarantee that the `invokeId` values are unique by combining `c` with a number from a generated sequence. This sequence of numbers is obtained from a pseudo random number generator, using a seed from the session key `Ks`, so that only the initiator and responder can be aware of the sequence. In fact both the initiator and the responder have their own sequence of numbers, since they will need to communicate asynchronously.

`i` = `f(n,c)` CMIP PDU sender
`c` = `g(i,n)` CMIP PDU receiver
`i` = `ROS invokeId`
`n` = one of the numbers from the generated sequence
`c` = CMIP PDU's checksum
`f` = the XOR function
`g` = $f^{-1}$ (Note: XOR and $XOR^{-1}$ are fast and effective)

#### *10.2.5.2.3 Data confidentiality*

DES encryption is utilised to prevent unauthorised entities from inspecting the contents of the CMIP PDUs. The CMIS PDU is encrypted and junk padding is then appended and prepended in order to prevent plain text attacks.

### 10.2.5.3 Access control

Access control for OSI management is based on the model described in ISO IS 10164-9 / ITU-T X.741 *Objects and Attributes for Access Control* SMF [10.17]. This function allows restrictions to be placed on a user's (the Initiator) right to access information (the Target) held at an agent, and it can be based on one or more of the access control schemes. Authenticated identities are a fundamental requirement for the usage of access control mechanisms.

#### *10.2.5.3.1 The access control model*

OSIMIS supports access control based on an *Access Control List* (ACL) scheme. The model is realised by a set of "Support Managed Objects" which identify:

- the default access control rules (the *accessControlRules* class);
- the specific access control rules (the *rule* class);
- the protected targets (the *targets* class);
- the initiators that are granted or denied access to targets (the *aclInitiators* class for ACL based schemes).

The inheritance hierarchy of those classes is presented in Figure 10.14.

The instances of those classes are combined through name bindings to form the containment schema shown in Figure 10.15. A single instance of the accessControl-Rules class is required per access decision function within a security domain, representing the default policy within that domain i.e. the application in an agent role whose managed objects are protected.

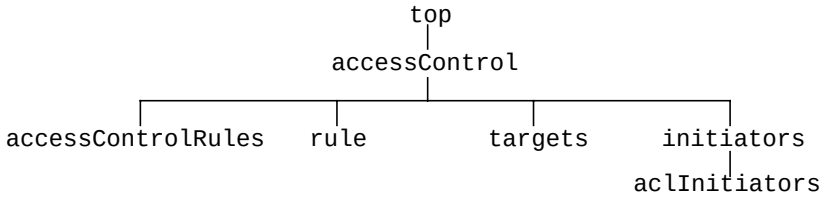



**Figure 10.14 Access control inheritance hierarchy**

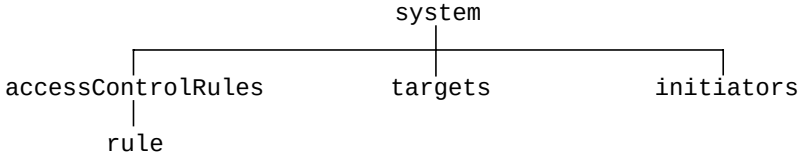



**Figure 10.15 Access control containment schema**

Containment is not the only relationship between these managed objects:

- rules may point to initiators and targets, binding them in order to grant or to deny access;
- targets may point to other managed objects that constitute the actual protected targets.

These relationships are realised through special attributes of those classes. We will now describe each of these classes and their attributes.

The *aclInitiators* class has one significant attribute, *accessControlList*, which is a list of distinguished names representing the initiators referenced under a security policy. Initiators may be bound to targets through item rules specifically, or otherwise they are either granted or denied access to everything through global rules.

The *targets* class identifies classes and instances for object level access control. Classes are identified through the *managedObjectClasses* attribute which is an object list. Instances are subjected through the *managedObjectInstances*, *scope* and *filter* attributes. The first attribute is a list of distinguished names identifying the target instances, whilst if scope and filter have values other than the defaults, they are applied to all the previous instances to yield the overall set of target objects. If both classes and instances are present, a boolean OR relationship is assumed.

The operations which are to be granted or denied for the identified target objects are specified through the *operationsList* attribute. The full list of operations are get, scope, filter, set, action, create and delete. The first three are 'monitoring' operations while the last four are 'control' ones. Scope means we allow for a target object to be selected through scoping whilst filter means we allow for it to evaluate a filtering assertion. Denying scope does not mean that the object in question is not visible at all, this depends on the enforcement action imposed by the relevant containing rule object.

The use of these facilities can provide very powerful capabilities to identify target objects in a dynamic fashion. Only some of their 'properties' need to be known in advance in order to be catered for by a policy based on static access control objects. In short, these facilities allow the definition of policies which will be valid for any objects created dynamically in the system either through the management interface or as a result of real resource interactions.

The *rule* object class binds initiators and targets through an enforcement action that may be allow (grant rule) or deny (deny rule). The attributes of the rule class are *initiatorsList*, *targetsList* and *enforcementAction*. The first two are both distinguished name lists pointing to the contained initiators and targets respectively. The *enforcementAction* attribute may take the following values: *denyWithResponse* (default), *denyWithoutResponse*, *denyWithFalseResponse*, *abortAssociation* or *allow*. If enforcementAction has any one of the first four values it is a deny rule while the last one, *allow*, renders it a grant rule. *DenyWithoutResponse* means that no reply is sent back to the initiator i.e. this facility is used to hide some targets completely. This does not mean that the initiator will possibly deadlock: the CMIS/P rules are obeyed and an empty reply will terminate a sequence of linked replies or a single request in this case.

If a rule has an empty targets list it is a global rule else it is an item rule. Global deny rules are used to refuse access to everything, whilst global grant rules grant access to everything for certain initiators. If the initiators list is empty, it means that the rule applies to all the initiators.

As the precedence of evaluation of rules supports a “least privileged” policy, the order of evaluation for any requested operation is:

1) global deny rules,
2) item deny rules,
3) global grant rules,
4) item grant rules,
5) default rules.

This order of evaluation essentially means that the strictest deny rules have precedence; whilst if there are no deny rules for the operation and target in question, the most “generous” grant rule applies.

Finally, the *accessControlRules* class identifies the default rules that should apply if no global or item rules have denied or granted access for the operation and target in question. The main attributes are *defaultAccess*, *defaultDenialResponse* and *denialGranularity*. DefaultAccess allows or denies access for each of the possible operations. If an operation is not present, the default value is deny. Note that deny and allow are expressed through the enforcementAction syntax, which means that four values may be used to express denial: Note that all of these mean simply deny in this context, it is the defaultDenialResponse attribute that determines the exact type of denial. This can have *denyWithResponse*, *denyWithoutReponse*, *denyWithFalseResponse* and *abortAssociation* values. Only the first two are currently supported. Finally, the denialGranularity attribute allows the level of denial to be controlled. Denial at the request, object and attribute levels are possible but the current implementation supports only object level denial.

#### 10.2.5.4 Concluding comments on the OSIMIS security mechanisms

We have set out to provide security solutions that balance a high level of security against any associated reduction in the timeliness of TMN management operations. In the absence of hardware support for cryptographic processing and key storage, the mechanisms described in this section are perceived to provide an optimal trade-off: Indeed extensive trials have confirmed that the DES based authentication, data integrity, PDU ordering confirmation and access control security mechanisms provide the TMN with a very strong armoury in the battle against the onslaught of both passive and active attacks.

### 10.2.6 Directory support: shared management knowledge and location transparency

A TMN is a highly distributed environment in which each component management process needs access to information on the location, the offered services and the supported managed objects of the other management processes; this information is collectively termed *Shared Management Knowledge* [10.18]. The OSI Directory Service [10.4] provides the means for storing the SMK, since it provides a multi-purpose globally distributed database system.

The Directory Service model structures information about a large number of objects, such as services, network resources, organisations and people, in a hierarchical

fashion termed the *Directory Information Tree* (DIT), which is the X.500 equivalent of OSI management's MIT. Information objects can be created and deleted, and have their attributes accessed (read or written). Access can involve complex assertions through filtering as with CMIP. This object-oriented information store is distributed over physically separate entities known as *Directory Service Agents* (DSAs). These communicate with each other through a special protocol and any requests for information that a 'local' DSA does not hold can be 'chained' to all the other DSAs until the information is found. A management process can access this information via *Directory User Agents* (DUAs) which communicate with the 'local' domain DSA using the *Directory Access Protocol* (DAP).

The principal reasons for selecting the Directory as the global SMK repository are:

- It provides a global schema for naming and storing information about objects that are highly distributed. For example, every management process in the world can be registered with a unique name, i.e. its Distinguished Name (DN).
- It provides powerful mechanisms for transparently accessing this global information (e.g. searching within some scope in the DIT using some filter).
- One of the major objectives of the OSI Directory, since it was recommended, was to provide an information repository for OSI application processes. For example, by keeping the locations (i.e. OSI presentation addresses) of the various application entities representing the application processes within the OSI environment.

#### 10.2.6.1 Globally unique naming using Distinguished Names

Since both the MIT and DIT naming schemes are hierarchically structured, by 'splicing' them together we can give a globally unique DN name to every Managed Object. This is achieved by taking a given MO's Local DN sequence, which is unique with respect to a given *Systems Management Application Process* (SMAP), and appending this to the DN of the SMAP itself. For example, from Figure 10.16 below we can see that the high-lighted *SwitchX* Q-Adaptor managed object, has a locally unique naming sequence of:

```
{systemId=switchX@ifDir=output@ifId=3}
```

giving a combined globally unique name of:

```
{c=GR@o=FORTH@ou=ICS@cn=ManagementProcesses@cn=SwitchX
-QA@systemId=switchX@ifDir=output@ifId=3}
```

#### 10.2.6.2 Providing location transparency

Location transparency is a basic mechanism in a distributed environment (ITU X.900). In the TMN, it provides a means for finding the address of SMAPs in a location independent way. Bearing in mind that the location of a SMAP may change over time (e.g. a Q-Adaptor for some ATM-switch that is running on machine X might migrate to some other machine if X crashes), we conclude that location transparency should be supported in a TMN. Since the location of a SMAP does not change very frequently, the OSI Directory is appropriate for storing, retrieving and modifying location information for SMAPs.

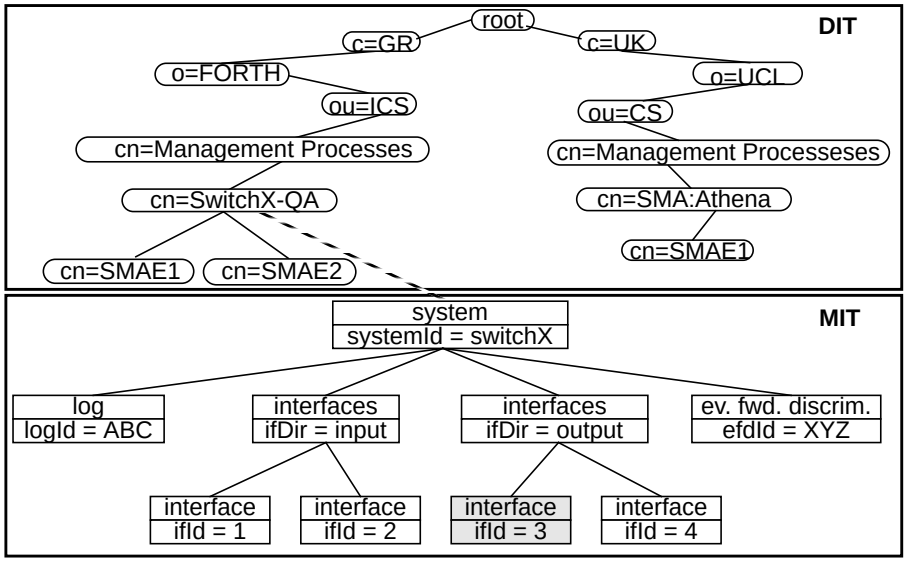



**Figure 10.16 Global management application and object naming**

The basic requirement for a location transparency mechanism is that, given a SMAP's name, i.e. its *Application Entity Title* (AET), it should provide a means of identifying the location, i.e. the OSI presentation address (PSAP) where the *Systems Management Application Entity (*SMAE) representing that SMAP is awaiting, either for management operations or notifications. In the TMN though, there is the possibility that a SMAP is represented from more than one SMAE, for example, consider the following cases:

- *SwitchX-QA* depicted in Figure 10.16 above, that is replicated so as to achieve fault tolerance,
- a Network Element Level Manager (NELM) that can act as a manager (by issuing management operations to lower level processes) and an agent (by serving management requests issued by a NLM-OS) at the same time,
- a SMAP that supports more than one interoperable interface, meaning that a different SMAE might be present for each interface,
- a SMAP that provides some management service can implement a number of management functions. These management functions will be provided by a number of SMAEs representing the SMAP.

Bearing these in mind, a location transparency mechanism involves choosing among a number of SMAEs representing the SMAP we wish to communicate with. In order to provide this functionality, the following information should be kept in every directory object that represents an SMAE:

- the application context supported from the communicating entity,
- the presentation address (PSAP) where this SMAE is located,

- Additionally, every SMAE directory object should contain information regarding the systems management application service element (SMASE) and the common management information service element (CMISE) in the SMAE. This information includes the supported systems management application service (SMAS), functional units (FUs), the supported management profiles, the supported CMIP version and the supported CMIS FUs on every SMAE.

In our current implementation, every SMAP has the ability to update (by issuing a DAP modify, add or remove operation) the directory objects that represents itself and its corresponding SMAEs. These update operations take place on the start-up and shutdown of a SMAP. Having the above information about each SMAE registered in the Directory, each SMAP (either in the manager or agent role) can establish an association with any other named SMAP, in a location transparent way by performing the following step:

- Given the DN of the SMAP it wishes to associate with, it performs a DAP search under the following conditions (i.e. we use a filter with the following conditions):
  - the DN of the SMAP is used as the base object for the search,
  - search for objects with the standard application context name '*systems-management*',
  - search for objects that support the interoperable interface through which it wishes to communicate (by checking the supported CMIP version and the supported CMIS FUs),
  - search for objects that perform a specific management function in the opposite role (by checking the supported SMAS FUs and the supported management profiles),

which should return the value of the required presentation address attribute value of the matching SMAE.

## 10.3 The OSIMIS TMN development process

### 10.3.1 The development phases

The Guidelines for the Definition of Managed Objects [10.10], provide a formal means of defining Managed Objects (MOs). Essentially, it is an object-oriented specification language that is used to define a set of Managed Object Classes (MOCs), which at runtime are instantiated as MOs. For each MOC, its position is specified in both the inheritance and containment hierarchies of a MIB, the names and types of its attributes, the notifications that it can emit, and the actions that can be performed on it.

In theory a GDMO compiler should be able to take a MIB specification and produce a management agent that implements the MIB; in practice, the process is more involved due to the fact that GDMO defines the required behaviour of each MOC in natural language, which can not be readily parsed. In addition certain MOC attribute types may not have been pre-installed into OSIMIS, though utilisation of the provided ASN.1 and attribute compilers minimises the task of generating a C++ class for each new attribute type.

Each MOC in OSIMIS is represented by a C++ class definition, which at run-time is instantiated as an MO. The OSIMIS GDMO compiler therefore has to generate OSIMIS MO compliant C++ header and methods files. The attribute compiler will generate the necessary C++ files for any additional attribute types. This leaves the provision of the code for MOC specific behaviour and the interface to the real resources, which must be provided manually.

### 10.3.2 The ASN.1 and OSIMIS attribute compilers

For each MOC attribute type's syntax, which is specified in ASN.1, the existing ISODE *Pepsy* compiler can produce a C language structure, together with functions for encoding and decoding, printing, string parsing and performing comparisons. High-level C++ abstractions are then used to provide the support for dealing with these ASN.1 based attribute types, in such a way that utilisation of the actual abstract and transfer syntaxes of the ISODE ASN.1 support service are hidden from the implementer by *encapsulating* the Pepsy compiler output.

In ICM an ASN.1 Attribute compiler has been designed and implemented, which can automate the production of the required C++ attribute classes. The only minor problem with this approach is that the automated mechanisms for the print and parse methods may utilise string formats that the agent implementor might wish to replace with their own preferred format; whilst sometimes the automatically produced compare method may be incorrect due to buried-in semantics. In these cases hand-coded methods can be provided which simply over-write the automatically produced ones.

### 10.3.3 A platform-independent GDMO compiler

Being script driven, the OSIMIS GDMO compiler is completely platform-independent i.e. it does not contain any hard-coded knowledge of a specific network management platform, in our case of OSIMIS. Typically a compiler parses a program (of GDMO statements, in this case) and builds up a symbol table representation of the program; in our compiler the symbol table is represented as a set of C++ objects. The code generation phase of the compiler then produces code for the target platform. In our compiler, the code generation phase is controlled by an interpreted script language. The script language is able to address the information in the compiler's symbol table and to output the information to files that conform to the requirements of a particular network management platform; in software engineering terms, this is achieved by adding meta class information to the C++ symbol table objects so that they can be used as variables in the script language. In short, the symbol table data of the compiler is entirely malleable. It is in this sense that the knowledge of a particular network management platform is not hard-coded in the GDMO compiler.

Compiler scripts supplied with OSIMIS, permit the extraction of information from the compiler's symbol table, leading to the generation of OSIMIS conformant MO header and methods files. The script language also has primitives that make it possible to merge hand written code and compiler generated code.

*Gdmoasn-cmpl* is a Unix shell script that invokes three programs that generate the code that is linked into an agent executable. A modified version of the ISODE Pepsy

ASN.1 compiler parses the ASN.1 module and generates C encoders and decoders; the OSIMIS Attribute compiler then takes the code generated by Pepsy and encapsulates it into OSIMIS conformant C++ attribute classes. The GDMO compiler parses the GDMO script and generates a C++ header and methods file for each managed object class defined in the GDMO script; at the same time it includes the hand written code that manages the real resource. Finally, it generates a Unix Makefile that will compile the managed object C++ files and generate a library that can be linked to the OSIMIS libraries (such as the GMS and kernel libraries) to create an agent. Figure 10.17 below illustrates the sequence in which the various programs are executed.

The GDMO compiler uses two databases to resolve references, a database of external attributes which resolves GDMO attributes that are DERIVED FROM externally defined attributes and a database of syntaxes that resolves all the WITH...SYNTAX clauses in the GDMO script. Both databases have the same format, the GDMO attribute or syntax name followed by the name of its corresponding C++ class and the name of the header file that defines the C++ class. The contents of these databases are loaded into the compiler's symbol table at run-time; the OSIMIS scripts can then generate code that resolves all the references.

As it can be seen from the Figure 10.17, the *asn-cmpl* script initially reads the GDMO module and a syntax database that contains a list of those syntaxes provided by OSIMIS so that a list of undefined attribute types can be generated by the Pepsy and Attribute compilers. The GDMO compiler then updates its symbol table with the newly defined attribute types.

When the GDMO compiler generates code, it has to include hand written code that manages the real resource. The following conventions have been adopted and programmed into the scripts that control the compiler's code generation. For each MOC, the GDMO compiler looks for two files of hand written code that are merged with the compiler generated code. One file, whose name is formed by concatenating the name of the MOC with '`.inc.h`', should contain user defined data and method declarations. Another file, whose name is formed by concatenating the name of the MOC with '`.inc.cc`', should contain user defined methods. If an MO does not require any user defined methods or data, there is no need to create these files. The hand written files are included using the C `#include` pre-processor directive. The compiler sets up the relevant dependencies in the Makefile so that if one of the hand written files is changed, only the relevant C++ class will be recompiled.

The hand written code has to take into account the object-oriented nature of the OSIMIS platform. At least one of the methods of hand written code would be expected to redefine a C++ virtual method in the OSIMIS GMS; for example, you might want to redefine the virtual method `get`, which resolves CMIP M-GET requests, so that when the request is received, the values of the MO's attributes are refreshed by an appropriate call to the real resource.

In the long run, the large amount of initial effort that has been invested in building a platform-independent GDMO compiler should pay dividends. OSIMIS is a research platform that is continually evolving and thus will demand changes in the GDMO compiler; it will be much easier to modify a set of interpreted scripts than a large amount of complex C++ code.

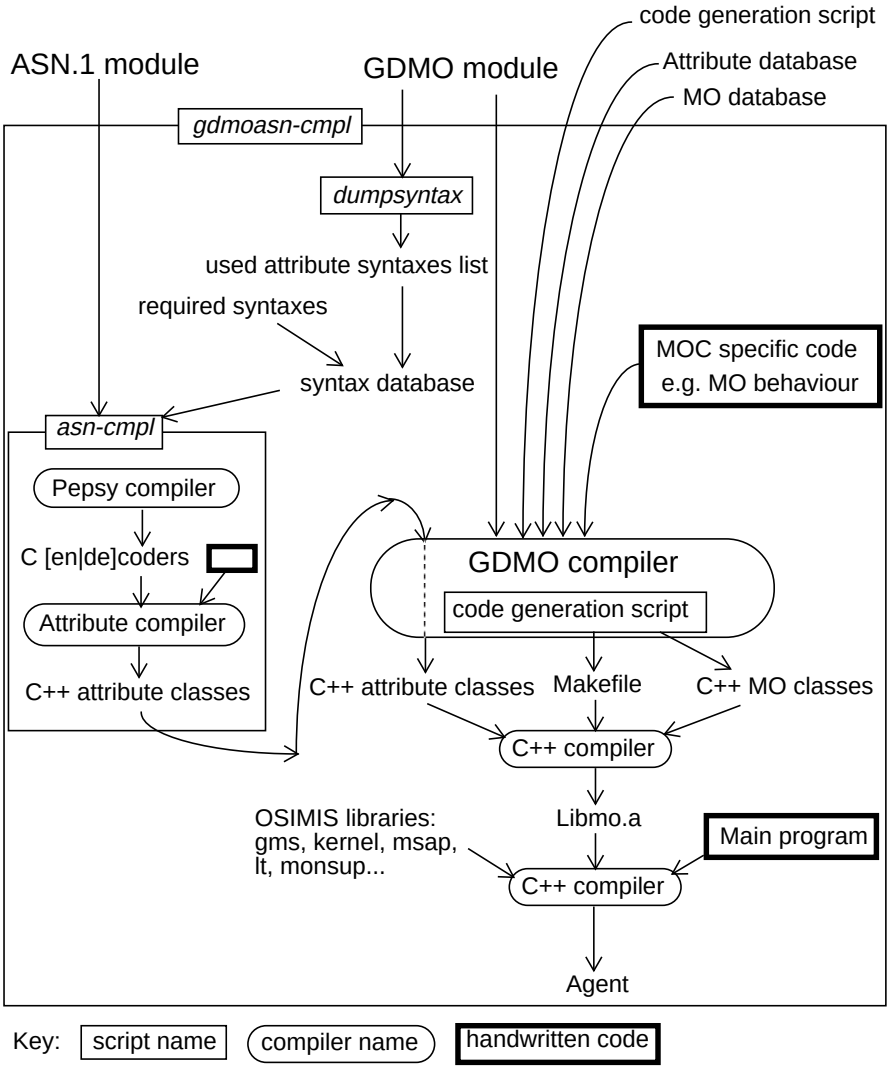



**Figure 10.17 TMN agent generation sequence**

## 10.3.4 Concluding remark on the TMN development process

When the Attribute and GDMO compilers are considered and the array of built-in SMFs, distribution services and security mechanisms are taken into account, it is our contention that the OSIMIS platform goes a very long way down the road towards the fully automatic creation of TMN management processes.

## 10.4 The OSIMIS applications

OSIMIS includes two types of generic management applications: semantic-free managers 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 SNMP. A very recent advance has been the provision of a meta-management system that manages the TMN applications themselves.

### 10.4.1 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 management operations to be applied and also provides a monitoring facility. Its successor, which is to be re-engineered in Tcl/Tk, will also have the capability of receiving event reports and of monitoring objects through event reporting.

In addition OSIMIS provides 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, actualClass and scoping) and the manipulation of the ANY DEFINED BY ASN.1 syntax through the table driven approach described in Section 10.2.1.

The integration of the platform’s location transparency and lightweight security mechanisms provide these management applications with secure access to location independent resources.

### 10.4.2 Sample agent

OSIMIS contains a non-standard implementation of an OSI Transport Protocol (TP) MIB. This manages the TP implementation of the ISODE stack and has proved to be very useful in monitoring the activity of ISODE based applications, such as DSAs, MTAs, transport bridges and even management applications themselves.

### 10.4.3 Distributed processing example

To demonstrate the use of the OSI management model as a powerful paradigm for distributed processing (see Section 10.5.1) when supported by high-level object-oriented APIs, the OSIMIS platform was extended with a distributed processing example. A *simpleStats* class was specified providing simple statistical services in the form of actions. The currently supported services (actions) are:

- *calcSqrt* - returns the square root of a non-negative real number

- *calcMeanStdDev* - calculates and returns the mean and standard deviation of a series of real numbers

The generic *maction* utility can be deployed to provide a 1 line shell script (!) that provides access to these services. In case a more complex client program needs to utilise these services, the RMIB access API may be used; a demonstration client is included which required only 20 lines in C++. The OSIMIS location transparency service may be used to hide the location where an instance of *simpleStats* object executes.

## 10.4.4 Management domain interoperability

Whether driven by technological merit, simplicity of development or government profiles, considerable investments have been made and will continue to be made into the provision of network management solutions based on the two dominant management architectures, namely SNMPv1 [10.27] and OSI [10.5]. They exist together so they must be made to coexist, so as to achieve global inter-working across heterogeneous platforms in the management domain. It is the authors' contention that coexistence can most readily be achieved by selecting a semantically rich reference model as the basis for this inter-working. Such an approach can then be readily extended to encompass both up and coming technologies such as CORBA [10.21] as well as future architectures.

The collaborative work of the Network Management Forum's (NMF) ISO/CCITT and Internet Management Coexistence (IIMC) group has provided a sound basis to our efforts in achieving coexistence through automated application level gateways. Through out this section we shall use the terms 'proxy', 'application level gateway' and 'Q-Adaptor' [10.3] synonymously, to indicate the automated translation of information and protocol models, so as to achieve the representation of management objects defined under one proprietary paradigm under that of an alternative model, namely OSI.

### 10.4.4.1 Information model mapping

#### *10.4.4.1.1 Overview*

Although conceptually similar from a very high level, the SNMP and OSI Management Frameworks are based upon very different approaches to the problem of open network management. One area where this difference is glaringly obvious is in their approaches to formalized descriptions of management information bases (MIBs).

The approach of the SNMP structure of managed information (SMI) is a very straightforward extension to Abstract Syntax Notation One (ASN.1), and uses a simple object-*based* approach to describing managed objects, defined using a small set of ASN.1 macros. These objects tend to be either very self-contained and atomic, or else are simple tables of other objects. This leads to a very straightforward MIB structure and approach to naming of MOs, describing simple objects, each with a number of useful attributes, or tables of simple objects.

In the OSI world, a completely object-*oriented* approach is prescribed by the Guidelines for Definitions of Managed Objects (GDMO) [10.10]. This approach is

very powerful, as it allows for reusable definitions of Managed Object Classes, Attribute Classes, Notifications, Behaviours, etc. The structure of the management information tree (MIT), is very flexible and is based on the use of specialized Naming Attributes to create very complex trees of information.

The need for automated conversion of SNMP MIB descriptions to the OSI GDMO format and vice-versa is a requirement for open-minded people interested in peaceful coexistence of both the Internet and ISO management frameworks, in order to best leverage the power of them both. In fact, the "Internet ISO Management Co-existence" (IIMC) work included the creation of practical solutions to allow this. The work presented on IMIBTOOL in this section is based on and contributed to the IIMC process [10.23][10.25] during 1993-94.

#### *10.4.4.1.2 Introduction to IMIBTOOL*

The purpose of IMIBTOOL is to completely automate the conversion of SNMP MIB descriptions to GDMO MIB descriptions (see Figure 10.18). Most of the rules for doing this have been described in [10.23]. Such conversion is a tedious, error-prone and repetitive task if done by hand; so in order to free the programmer to concentrate on more challenging tasks, the IMIBTOOL was developed.

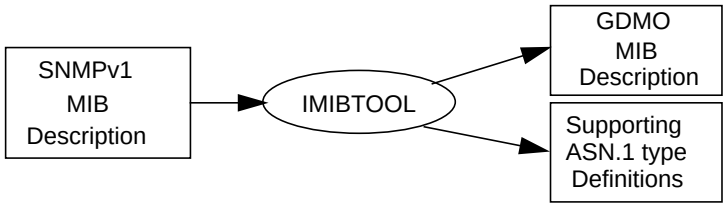



**Figure 10.18 MIB description conversion using the IMIBTOOL**

The IMIBTOOL was based on the popular SMIC (SNMP MIB Compiler) program [10.28]. This program is basically the front-end for an SNMPv1 MIB compiler[1]. It has many useful options, including generation of various types of useful information from an SNMPv1 MIB description. Basically, it syntactically checks an SNMPv1 MIB description, builds a very rich parse tree, and generates various types of outputs based on the command line options. It was decided to take this existing MIB compiler and add an option to convert the parse tree into GDMO plus ASN.1 output based on the IIMCIMIBTRANS [10.23] rules. This turned out to be a very fast, effective solution. During the course of writing IMIBTOOL, a number of inputs where made to the IIMCIMIBTRANS conversion rules that would not have been determined otherwise. Imibtool is the first openly-available translator for this purpose.

#### *10.4.4.1.3 The IMIBTOOL translation process*

Figure 10.19 shows the `imibtool`'s input and post-processing requirements. The include file specifies all the SNMPv1 MIB definitions for a complete MIB, but one

1. The next generation of the SMIC compiler also supports SNMPv2.

may only want to convert some but not all of the sub-MIBs. `Imibtool` may also generate some supporting ASN.1 definitions, as explained in the IIMC documents.

One annoying aspect is that there is no way to automatically identify SNMPv1 group objects. The only way for `imibtool` to recognize them for SNMPv1 is to have a built-in table of Object-Identifiers of SNMPv1 group objects, though in the future this will be handled via a configuration file.

The IIMC documents also define certain keywords held in GDMO behaviour clauses, that describe aspects such as how MOs can be created or deleted. It is not possible to automatically generate the correct CREATE/DELETE parameters, such as 'CREATE WITH-AUTOMATIC-INSTANCE-NAMING', and so some degree of post-processing is required, via a "sed" script.

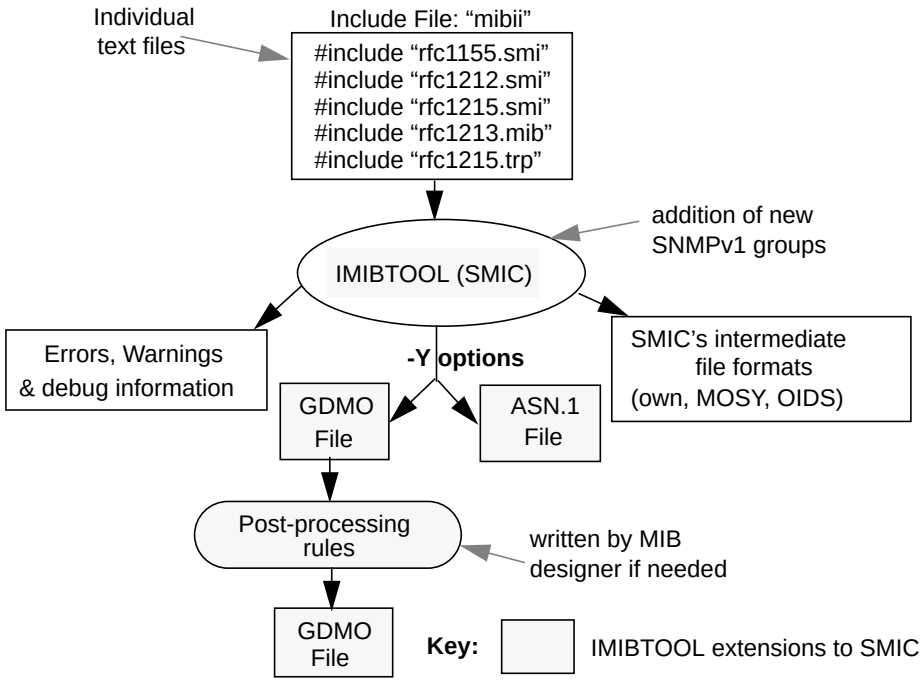



**Figure 10.19 Using IMIBTOOL**

`Imibtool` has proven to be a very useful tool for automated conversion of SNMPv1 to GDMO MIB descriptions. It would be very nice to extend imibtool to also handle SNMPv2 extensions as well, this would eliminate the need to identify SNMPv1 groups. `Imibtool` has been used by many people around the world wishing to automatically convert their SNMPv1 MIBs.

### 10.4.4.2 The generic CMIS/P to SNMP gateway

The current industry standard for network element management is the Internet SNMP, which is less powerful than 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/P 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 ICM project contributed actively to them and also built a generic application gateway.

This work involves a translator between Internet MIBs to equivalent GDMO ones (i.e. the `imibtool`) and a special script for the GDMO compiler which produces runtime 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 SNMPv1 but may be extended to cover SNMPv2.

A fundamental design requirement for the gateway is to achieve seamless interoperability between TMN management Operations Systems (OS) or Mediation Functions (MF) and SNMPv1 managed resources. An efficient mapping is essential given the fact that the gateway introduces an intermediate hop in the manager/agent communication path, see Figure 10.20.

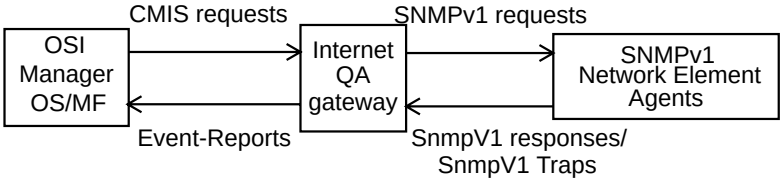



**Figure 10.20 Manager/agent communication paths**

#### *10.4.4.2.1 The Internet Q-Adaptor (IQA) gateway in operation*

An underlying aim of our research is to maximise the level of automation in generating a Q-Adaptor that proxies for the desired remote SNMPv1 agents. Three stages are required, namely *translate*, *convert* and *run*, see Figure 10.21.

- Translation involves the usage of the OSIMIS SMI to GDMO converter ('`imibtool`') to produce an OSI representation of the MIB that is to be managed.
- Conversion yields a simplified MIB description using the GDMO compiler.
- Run - the gateway reads in the simplified input file MIB description(s) and is ready to provide an 'OSI view' of the SNMPv1 managed real resources.

### 10.4.5 Managing the TMN itself

Whereas the scope of the TMN is to manage networks and services, the scope of metamanagement is to manage the TMN itself. We introduce the term *metamanagement* to refer to the *function of managing the management processes, systems and software comprising the TMN*. This includes the start-up and shutdown of management processes, recovery procedures in case of failures, as well as the automatic distribution and update of code and configuration files, so that an easily configured and fault tolerant TMN is provided. The metamanagement service has yet to be formally defined in the

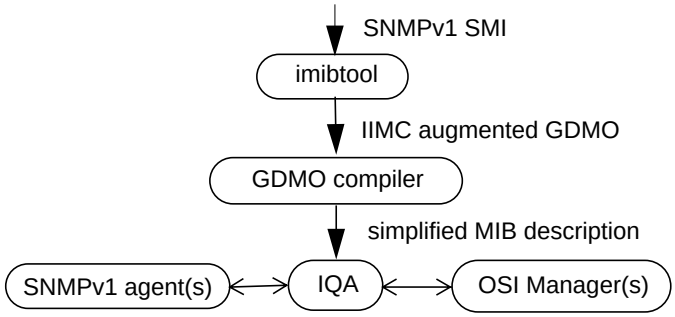



**Figure 10.21 The Internet Q-Adaptor's execution cycle.**

standards; so the OSIMIS meta-management service is undergoing a continuous evolutionary process, where we implement many features that have proved useful in ICM.

Within this section we will assume that the various TMN management processes interact according to the enhanced OSI manager/agent model proposed in [10.37]. This means that each management process is capable of not only updating the Directory with information about its state, functionality, location etc., but can also interrogate the same information about other processes via Directory queries. This provides the critical information about which processes exist in a TMN, the management services they provide and a location transparent way for associating with them. This kind of information, also called Shared Management Knowledge, forms the basis for our metamanagement system which is able to maintain a global view of the various TMN building blocks and the way they interact.

The TMN building blocks (OS, MDs, QAs, NEs and WSs) are installed on a network of computers which communicate over the DCN. In this section, we call these computers *TMN hosts*. The only requirement for a TMN host is to run a multitasking operating system which uses OSI to communicate with other hosts in the TMN.

We offer a tool to the TMN operator that helps him configure the TMN. After the configuration, the metamanagement system monitors the operation of the TMN and informs the operator of any exceptional conditions. It is the responsibility of the operator to take appropriate action. This initial 'manual' mode of operation provides the basic capabilities of metamanagement, and permits an evolutionary path towards its eventual automation by incorporating intelligence into the metamanagement components themselves.

Figure 10.22 depicts the metamanagement system model. Two new processes are introduced: the *Meta-Management Agent* (MMA) and the *Meta-Management Operations System* (MMOS). It is assumed that every TMN host runs a MMA acting in the agent role containing a MIB presenting managed objects representing the management processes running on that TMN host. Additionally, the MMA contains an FTAM client that is capable of connecting to an appropriate FTAM server in order to retrieve software and configuration files not present locally on the TMN host.

The MMOS acts in the manager role. It implements an OS equipped with an HCI (Human Computer Interface) through which the TMN operator is capable of perform-

ing appropriate management operations on the MMAs and therefore the management processes on the TMN hosts. Both the MMA and the MMOS contain a Directory User Agent (DUA) that performs the Directory updates and queries described in [10.37].

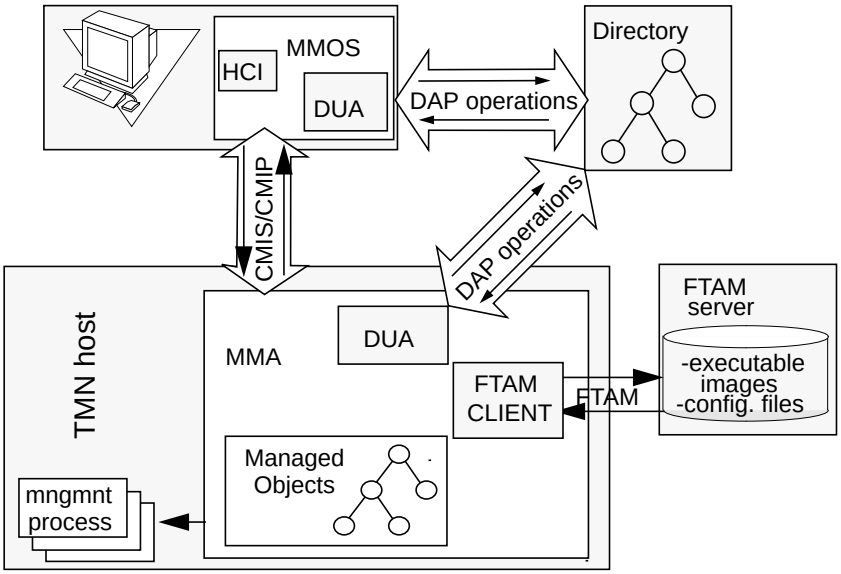



**Figure 10.22 The OSIMIS metamanagement model**

The MMA and MMOS are critical for the operation of the TMN and are therefore implemented as fault tolerant processes. The MMOS obtains information concerning the running MMAs, and therefore the available TMN hosts, through Directory search operations. Additionally it keeps the current load of each TMN host and an estimation of the load of the DCN links interconnecting the MMOS to that particular MMA.

The TMN operator, using the MMOS HCI, determines the appropriate TMN host on which to run a new management process. The new management process is started via a CMIP Create operation from the MMOS to the chosen MMA, i.e. the MMOS creates a managed object within the MMA's MIB.

The MMA reads from the Directory the entry corresponding to the process's software package and checks whether the necessary files are available locally. If the software package is already available, the MMA executes the process and updates its MIB with the newly created MO. After receiving the appropriate object creation message the MMOS updates the Directory with information about the started process. This information includes the MMA that is responsible for the process, the software package that the process runs and the execution vector used to start the process. If the software package is not available on the TMN host (or it is out-of-date), the MMA retrieves the files from the TMN FTAM server located in one of the TMN hosts. After retrieving the necessary files the MMA proceeds as above.

As well as creating new management processes, the TMN operator may need to terminate currently running management processes, e.g. during reconfiguration of the

TMN. To terminate a management process, the MMOS issues a CMIP Delete operation on the appropriate managed object in the respective MMA.

Each MMA periodically checks whether the management processes running on its TMN host are alive (e.g. by issuing a CMIP Get request for the `system` MO). In case of a process failure, the MMA sends an appropriate notification to the MMOS. The TMN operator (through the HCI of the MMOS) is always able to access a complete view of the TMN. He can query the directory for TMN hosts, the processes that run or are available, where they run and their operational status.

We have shown that a management model based on OSI systems management concepts is rich enough to support the requirements of a metamanagement system. This type of management is typical in distributed systems contexts but we have shown that the OSI management and directory models provide a very good solution in an OSI management environment.

## 10.5 Other areas of application for OSIMIS

Up to now it has been assumed that OSIMIS will be used purely as a TMN management platform. Though this is the reason it was conceived, 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.

### 10.5.1 OSIMIS as a general distributed systems platform

All the facilities described above make OSIMIS suitable as a general vehicle for building distributed systems. The CMIS Action primitive is essentially a general object method i.e. a remote operation, now the combination of remote operations, location transparency and high-level object access mechanisms make it 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 very high and experiments have shown that it is directly comparable, even for simple information access, to that of SNMP-based ones.

### 10.5.2 OSIMIS as an object-oriented distributed database

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 manipulated through the management protocol - standard eventLo-

gRecords 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. At the moment OSIMIS does not yet provide transaction support, which is very important for such usage. When these are implemented, the existing infrastructure will make it possible to use OSIMIS for storing information in a distributed fashion while providing sophisticated authentication and access control mechanisms. It should be noted that transaction services are particularly useful in the higher TMN layers where more static information e.g. service, and customer specific data, is stored.

## 10.6 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 what has been provided until now by most commercial offerings. In complex hierarchical management environments, as epitomised by a TMN, object-oriented agent support similar to that of the GMS and the associated tools and functions is fundamental, as is 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.

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, whilst a third one, the ODP-based OMG CORBA framework threatens to become a *de facto* industrial standard. All 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. OSI is the most powerful one in terms of both protocol operations and information model expressiveness, justifying as such its choice as the basis of OSIMIS.

## 10.7 Acknowledgements

We would like to thank Graham Knight and Saleem Bhatti, from the ESPRIT MIDAS project, for their contributions to OSIMIS. Our colleagues from international collaborative endeavours such as the NMF's IIMC also deserve a special mention.

## 10.8 References

[10.1] B.Stroustrup, “The C++ Programming Language,” Addison-Wesley, Reading, MA, 1986.
[10.2] B.W.Kernigham, D.M.Ritchie, “The C Programming Language,” Prentice-Hall, New Jersey, 1978.
[10.3] ITU-T M.3010, Principles for a Telecommunications Management Network, Working Party IV, Report 28, Dec. 1991.
[10.4] ITU-T X.500, Information Processing - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Service, 1988.
[10.5] ITU-T X.701, Information Technology - Open Systems Interconnection - Systems Management Overview, July 1991.
[10.6] ITU-T X.710, Information Technology - Open Systems Interconnection - Common Management Information Service Definition, Version 2, July 1991.
[10.7] ITU-T X.711, Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification, Version 2, 7/91.
[10.8] ITU-T X.720, Information Technology - Open Systems Interconnection - Structure of Management Information - Part 1: Management Information Model, January 1992.
[10.9] ITU-T X.721, Information Technology - Open Systems Interconnection - Structure of Management Information - Part 2: Definition of Management Information, February 1992.
[10.10] ITU-T X.722, Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects, August 1991.
[10.11] ITU-T X.730, Information Technology - Open Systems Interconnection - Systems Management: Object Management Function, January 1992.
[10.12] ITU-T X.733, Information Technology - Open Systems Interconnection - Systems Management: Alarm Reporting Function, February 1992.
[10.13] ITU-T X.734, Information Technology - Open Systems Interconnection - Systems Management: Event Management Function, February 1992.
[10.14] ITU-T X.735, Information Technology - Open Systems Interconnection - Systems Management: Log Control Function, September 1992.
[10.15] ITU-T X.738, Information Technology - Open Systems Interconnection - Systems Management: Metric Objects and Attributes, 1994.
[10.16] ITU-T X.739, Information Technology - Open Systems Interconnection - Systems Management: Summarisation Function, 1994.
[10.17] ITU-T X.741, Information Technology - Open Systems Interconnection - Systems Management: Objects and attributes for access control, 1995.
[10.18] ITU-T Draft X.750, Information Technology - Open Systems Interconnection - Systems Management: Management knowledge management function.
[10.19] ITU-T X.901, ODP Reference Model Part 1. Overview, 1995.
[10.20] Information Processing - Open Systems Interconnection - File Transfer, Access and Management, 1988.
[10.21] Object Management Group, The Common Object Request Broker Architecture and Specification (CORBA), 1991.

[10.22] X/Open, OSI-Abstract-Data Manipulation and Management Protocols Specification, January 1992.
[10.23] L.LaBarre (Editor), Forum 026 - Translation of Internet MIBs to ISO/CCITT GDMO MIBs, Issue 1.0, October 1993.
[10.24] April Chang (Editor), Forum 028 - ISO/CCITT to Internet Management Proxy, Issue 1.0, October 1993.
[10.25] L.LaBarre (Editor), Forum 029 - Translation of Internet MIB-II (RFC1213) to ISO/CCITT GDMO MIB, Issue 1.0, October 1993.
[10.26] D. Cass, M. Rose, ISO transport services on top of TCP, Network Working Group, Request For Comments 1006, May 1987.
[10.27] J.Case, M.Fedor, M.Schoffstall, J.Davin, A Simple Network Management Protocol (SNMP), Network Working Group, Request For Comments 1157, May 1990.
[10.28] SMIC was written by Dave Perkins of Synoptics, and is available in the ftp://ftp.synoptics.com/eng/mibcompiler directory.
[10.29] 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.
[10.30] J.P. Onions, “Transport Bridging,” Published in Interworking: Research and Experience, Vol. 1, pp. 27-34, John Wiley & Sons.
[10.31] G.Pavlou, T.Tin, “OSIMIS User Manual Version 1.0 for System Version 4.0,” Dept. of Computer Science, University College London, December 1994.
[10.32] G.Pavlou, “The OSIMIS TMN Platform: Support for Multiple Technology Integrated Management Systems,” Proceedings of the 1st RACE IS&N Conference, Paris, November 1993.
[10.33] T.Tin, G.Pavlou, R.Shi, “Tcl-MCMIS: Interpreted Management Access Facilities,” Proc. of the 6th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, Ottawa, Canada, October 1995.
[10.34] K.McCarthy, G.Pavlou, S.Bhatti, J.Neuman De Souza, “Exploiting the Power of OSI Management for the Control of SNMP-capable Resources Using Generic Application Level Gateways,” ISINM’95.
[10.35] National Institute of Standards and Technology, “Data Encryption Standard” FIPS Publication 46-1, January 1988.
[10.36] R. L. Rivest, A. Shamir, L. A. Adleman, “A Method for Obtaining Digital Signatures and Public Key Cryptosystems,” Communications of the ACM, 21 (2), pp. 120-126, February 1978.
[10.37] Stathopoulos C., Griffin D., Sartzetakis S.: “Handling the Distribution of Information in the TMN,” Proceedings of the fourth international symposium on integrated network management, ISINM’95, ed. Sethi A., et al., Chapman & Hall, 1995.
[10.38] RACE R2059 ICM internal deliverable, “Monitoring Management Function in OSIMIS, ICM/WP3/NTUA/0097,” Nov. 1994.